FACHHOCHSCHULE WEDEL

FACHBEREICH WIRTSCHAFTSINFORMATIK

Diplomarbeit

Entwicklung und Implementierung einer Rechercheunterstützung in der Lernumgebung LAssi

Eingereicht von:

Jan Christian Krause

Harburger Strasse 10, 21614 Buxtehude, Tel: (04161) 600 820

Abgegeben am:

22. Februar 2007

Erarbeitet im:

7. Semester

Referent:

Prof. Dr. Sebastian Iwanowski

Fachhochschule Wedel

Feldstraße 143

22880 Wedel

Tel: (04103) 8048 63

Betreuer:

Dr. Axel Schmolitzky

Universität Hamburg

Fachbereich Informatik

AB Softwaretechnik

Vogt-Kölln-Strasse 30

22527 Hamburg

Tel: (040) 42883 2302

Für Sandra

Inhaltsverzeichnis

Tabellenverzeichnis **6**

Abbildungsverzeichnis **7**

1 Vorwort **9**

2 Einführung **10**
2.1 Motivation 10
2.2 Ziele und Gang dieser Untersuchung 11

3 Analyse der Rechercheunterstützung von Schülern **12**
3.1 Schülerprobleme bei Recherchen im WWW 12
3.2 Lösungsansätze von Suchmaschinen 13
3.2.1 Vorschlagen alternativer Anfragen 14
3.2.2 Benutzereinfluss auf die Trefferanordnung 15
3.2.3 Benutzereinfluss auf die Trefferdarstellung 16
3.3 Fazit 17

4 Skizzierung des zu entwickelnden Systems **19**
4.1 Anforderungsdefinition 19
4.2 Entwicklung eines Lösungsansatzes 20
4.2.1 Abbildung einer Aufgabenstellung in der LAssi-Plattform 20
4.2.2 Herleitung des Lösungsansatzes 21
4.2.3 Entwicklung eines Benutzungsmodells 24
4.3 Veranschaulichung anhand eines Fallbeispiels 27
4.4 Abgrenzung zu verwandten Systemen 30

5 Grundlagen der Clusteranalyse **32**
5.1 Einführung in die Clusteranalyse 32
5.2 Ähnlichkeits- und Distanzmaße 33
5.3 Anwendung von Ähnlichkeits- und Distanzmaßen auf Cluster 36
5.4 Vorstellung des k-Means-Verfahrens 36

6 Statistische Bewertungsverfahren für Deskriptoren **39**
6.1 Absoluter Häufigkeitsansatz 40
6.2 tf-idf-Häufigkeitsansatz 41

7 Systementwurf **42**
7.1 Übersicht . . . 42
7.2 Bereitstellung von LAssi-Desktop- und Schaleninhalten . . . 43
7.3 Aufbereitung und statistische Analyse der Textinhalte . . . 46
7.4 Berechnung von Anfragen . . . 51
7.5 Interaktion mit integrierten Suchmaschinen . . . 56
7.5.1 Repräsentation einer Suchmaschine und eines Treffers . . . 56
7.5.2 Kommunikation mit Suchmaschinen . . . 58
7.5.3 Verwaltung der Treffer in Trefferschalen . . . 61
7.6 Entwurf der Clusterberechnung . . . 64

8 Implementierungsdetails **66**
8.1 Extraktion von Trefferinhalten . . . 66
8.2 Statusmonitoring der Suchagenten . . . 68
8.3 Implementierung der Trefferansichten . . . 71
8.3.1 Grafische Darstellung von Suchmaschinen und Treffern . . . 72
8.3.2 Implementierung der Herkunftsansicht . . . 74
8.3.3 Implementierung der Clusteransicht . . . 76

9 Systembewertung **79**
9.1 Vorstellung von Qualitätsmaßen . . . 79
9.2 Bewertung anhand eines Fallbeispiels . . . 80
9.3 Bewertung anhand eines Schuleinsatzes . . . 84
9.4 Funktionaler Vergleich mit einer Suchmaschine am Beispiel Google . . . 86
9.5 Fazit . . . 89

10 Abschlussbetrachtungen **91**
10.1 Anregungen für weitere Entwicklungen . . . 91
10.2 Zusammenfassung, Fazit und Ausblick . . . 93

A Benutzerhandbuch **94**
A.1 Ablaufbedingungen . . . 94
A.2 Programminstallation und -start . . . 94
A.3 Bedienungsanleitung . . . 94
A.3.1 Öffnen des Assistentenfensters . . . 95
A.3.2 Recherchen mit dem Assistenten . . . 96
A.3.3 Ansicht von Trefferschalen . . . 96
A.3.4 Arbeit mit Treffern und Steuern des Assistenten . . . 98
A.3.5 Anzeige des Recherchefortschritts . . . 99
A.3.6 Konfiguration des Assistenten . . . 99
A.4 Fehlermeldungen . . . 101
A.5 Wiederanlaufbedingungen . . . 101

B Protokolle der statistischen Untersuchungen **102**

C Integration einer Suchmaschine am Beispiel Yahoo **104**
C.1 Eclipse Erweiterungspunkte 104
C.2 Adapterklasse “YahooService” 105
C.3 Adapterklasse “YahooHit” 107
C.4 XSD-Spezifikation für Eclipse-Erweiterungspunkt 108

D Schülerfragebogen zur Bewertung der Assistentenleistung **110**

E Abkürzungsverzeichnis **112**

Literaturverzeichnis **113**

Eidesstattliche Erklärung **114**

Tabellenverzeichnis

7.1 Beispielhafte Tokenstatistik für zwei Karteikarten (KK) 48
7.2 Beispielhafte Kandidatenliste auf Basis der TF-Funktion 49
7.3 Anfragen aus Such- und Filterbegriffen zum Garnelen-Beispiel 55

Abbildungsverzeichnis

3.1 Rechtschreibprüfung der Suchmaschine Google 14
3.2 Vervollständigung einer Suchanfrage durch Google Suggest 14
3.3 Vorschläge für alternative Suchbegriffe von Web.de SmartSearch 15
3.4 Benutzeroberfläche von Yahoo Mindset 16
3.5 Grafische Darstellung der Treffercluster von Grokker 17

4.1 Modell eines unterstützten Rechercheprozesses für Schüler 22
4.2 Entwurf der Herkunftsansicht mit Treffern 28
4.3 Entwurf der Clusteransicht mit Treffern zur Anfrage “Garnele” 29
4.4 Entwurf der Herkunftsansicht mit Treffern zum Thema “Wattenmeer” 30

5.1 Auszug aus einer Merkmalsmatrix für das “Garnelen”-Beispiel 33
5.2 Cosinusmaß und Euklidische Distanz im zweidimensionalen Raum 35
5.3 Cluster mit zugehörigem Zentroiden im zweidimensionalen Raum 36
5.4 Das Prinzip des k-Means-Verfahren 37
5.5 Anwendung des k-Means-Verfahren zur Berechnung von Trefferclustern . . . 38

6.1 Auswahl von fünf Deskriptoren 39
6.2 Beispiele für LAssi-Karteikarten 40

7.1 Systementwurf für den Assistenten 42
7.2 UML Klassendiagramm für die Schnittstelle “IContentProvider” 44
7.3 UML Klassendiagramm für einen Iterator 44
7.4 UML Klassendiagramm der Schnittstelle “IContent” 45
7.5 Verarbeitungskette zur Generierung von Kandidaten aus Zeichenketten 46
7.6 Aufspaltung von Zeichenketten durch den Tokenizer 46
7.7 UML-Klassendiagramm der Klasse “DescriptorFactory’ 50
7.8 Datenstruktur des Parameters “tokenRelation” (“IRatorFunction”) 50
7.9 UML-Diagramm der Klasse “QueryService” 52
7.10 UML-Diagramm der Schnittstelle “SearchService” 56
7.11 UML-Diagramm der Klasse “Hit” 57
7.12 Allgemeiner Reflex-Agent und Suchagent 58
7.13 Schema einer Master-Slave-Architektur 59
7.14 UML-Diagramm der Klasse “AgentManager” und ihres Kontextes 60
7.15 UML-Diagramm der Klasse “HitService” und ihres Kontextes 62
7.16 UML-Diagramm der Klasse “ClusterGenerator” und ihres Kontextes 64

8.1 UML-Klassendiagramm der Klasse “DocumentReaderService” 67
8.2 Fortschrittsanzeige der Suchagenten 68
8.3 Fortschrittsanzeige der Suchagenten 69
8.4 Komponenten eines “ProgressComposite”-Objektes 70
8.5 Beziehung zwischen “AgentManager” und “AgentProgressView” 71
8.6 Trefferansichten des Assistenten 72
8.7 Trefferansichten mit dem JFace TreeViewer in UML-Notation 73
8.8 Beteiligte Klassen an der Berechnung und Darstellung der Herkunftsansicht . 75
8.9 Beteiligte Klassen an der Berechnung und Darstellung der Clusteransicht . . 76
8.10 Beispiel der Clusteransicht zum “Garnelen”-Beispiel 77

9.1 Gemittelte Messwerte für Fehlerrate und Precision 82
9.2 Gemittelte Messwerte für Fehlerrate und Precision 84
9.3 Treffer von Google zur Anfrage “Garnelen” 86
9.4 Rechercheergebnisse im ersten Durchgang des Assistenten 87
9.5 Rechercheergebnisse im zweiten Durchgang des Assistenten 88
9.6 Beispiel für den Inhalt eines LAssi-Desktops 89

10.1 Expertensuche im Klassenverband 93

A.1 Schaltfläche zum Öffnen des Hauptfensters 95
A.2 Hauptfenster des Assistenten 95
A.3 Benachrichtigung des Assistenten über abgeschlossene Recherche 97
A.4 Hauptfenster des Assistenten nach abgeschlossener Recherche 97
A.5 Clusteransicht des Assistenten nach abgeschlossener Recherche 98
A.6 Fortschrittanzeigen des Assistenten 99
A.7 Fehlermeldung des Assistenten 101

1 Vorwort

Ich möchte mich ganz herzlich bei Herrn Prof. Dr. Sebastian Iwanowski und bei Herrn Dr. Axel Schmolitzky für die Betreuung dieser Arbeit und die zahlreichen Anregungen bedanken.

Ebenso gilt mein Dank Herrn Michael Töpel und Herrn Michael Vallendor, die mir die Möglichkeit gegeben haben, diese Diplomarbeit im einzigartigen Umfeld des LAssi-Projektes zu entwickeln. Beide haben mich in vielen anregenden Diskussionen bei der Anfertigung dieser Arbeit sehr unterstützt.

Die zahlreichen Diskussionen mit meinen Entwickler-Kollegen Till Aust und Christian Späh haben mir viele wichtige Anregungen sowohl für die konzeptionelle als auch für die software-technische Umsetzung meiner Gedanken gegeben. Auch ihnen schulde ich dafür großen Dank.

Ohne die Unterstützung von Herrn Olaf Zeiske und seinen Schülern, insbesondere Vincent Fortuin, Serhat Etdöger, Mevlüt Vurmaz, Kjetil Fischer und Sinan Songül, wären die ersten Praxiserfahrungen mit meiner Software nicht möglich gewesen. Auf diesem Wege möchte ich Herrn Zeiske und seinen Schülern besonders danken.

Ganz besonderer Dank gilt meiner Freundin Sandra Dammann. Sie hat mich während meines gesamten Studiums selbstlos unterstützt und mir den Rücken freigehalten. Ohne sie wäre diese Diplomarbeit niemals entstanden. Ich danke außerdem meinem Vater Jan Peter Krause, der mir mein Studium mit ermöglicht hat.

Für die kritische Durchsicht meines Manuskriptes richte ich meinen Dank an Max Weichert, Florian Hauser und Fabian Runge.

Buxtehude, im Februar 2007

Jan Christian Krause

2 Einführung

2.1 Motivation

In den modernen Formen des Schulunterrichtes werden die Schülerinnen und Schüler in zunehmenden Maße mit der Aufgabe betraut, sich selbstständig in ein neues Wissensgebiet einzuarbeiten[1]. Sie sollen dadurch die Fähigkeit zum eigenverantwortlichen und selbstorganisierten Lernen erlangen. Dies wird als eine wichtige Kernkompetenz zur erfolgreichen Teilnahme und -habe an der Wissensgesellschaft angesehen. Das erworbene Wissen soll anschließend zielorientiert aufbereitet werden, z.B. für einen Vortrag vor der Klasse. Die Lehrerin oder der Lehrer tritt in diesen schülerzentrierten Unterrichtsformen stärker als motivierend, sowie begleitend und weniger als wissensvermittelnd in Erscheinung.

Das Projekt LAssi ("Learner's Assistant" oder "Lern-Assistent") ist aus einer Public-Private-Partnership zwischen der Freien und Hansestadt Hamburg und der IBM Deutschland GmbH hervorgegangen. Es verfolgt die Zielsetzung, eine indivduelle Lernumgebung zur Unterstützung von Wissensarbeit zu entwickeln. LAssi unterscheidet sich von E-Learning-Plattformen vor allem durch zwei Aspekte[2]:

- LAssi stellt Lernwerkzeuge bereit, keine Lernmaterialien

  LAssi entspricht einem Werkzeugkasten, in dem Werkzeuge zur Durchführung und Organisation von Wissensverarbeitung und -aneignung für Schüler bereitgestellt werden.

- LAssi ist eine indviduelle, dezentrale Plattform

  Jeder Schüler besitzt seinen eigenen Lern-Assistenten mit genau denjenigen Lernwerkzeugen, die ihn persönlich beim Lernen besonders unterstützen.

Ein zentraler Aspekt dieses schülerzentrierten Konzeptes ist die selbstständige Erschließung von Quellen und Wissen, z.B. in Form von Aufsätzen, Zeitungsartikeln, Webseiten oder Büchern, aus denen von den Schülern für die eigenen Lernzwecke Wissen extrahiert und aufbereitet werden kann. Daher wäre es ein großer Mehrwert für einen Schüler, wenn er mit

[1] Zukünftig wird der Einfachheit halber nur noch die männliche Form "Schüler" verwendet. Gemeint sind natürlich immer Schülerinnen und Schüler.

[2] Für weitere Informationen zum LAssi-Projekt wird auf dessen Internetseite http://www.lassitools.org verwiesen.

einem Werkzeug seines LAssi-Werkzeugkastens in der Lage wäre, sich eine Informationsgrundlage zu seiner aktuellen Aufgabenstellung zu beschaffen.

Die Entwicklung eines solchen Werkzeuges innerhalb des LAssi-Systems stellt aus drei Gründen eine besondere Herausforderung dar:

1. Das Werkzeug muss in der Lage sein, auf beliebig struktierten oder gar unstrukturierten Datenmengen zu operieren, weil die Schüler die unterschiedlichsten Suchräume für Recherchen heranziehen.

2. Das Werkzeug muss den Schüler individuell, also entsprechend seines Kenntnisstandes, unterstützen.

3. Das Werkzeug muss den Schüler kontextbezogen, also zu einer bestimmten Aufgabenstellung, bei der Recherche unterstützen.

2.2 Ziele und Gang dieser Untersuchung

Im Rahmen dieser Diplomarbeit soll eine Software konzipiert und in die LAssi-Plattform implementiert werden, welche die Schüler beim Auffinden von relevanten, digital verfügbaren Lernmaterialien und weiterführenden Quellen zu einer gegebenen, unterrichtsbezogenen Aufgabenstellung unterstützt.

Zur Erreichung dieses Zieles wird die aktuell alltägliche Situation von Schülern bei der Materialien- und Quellenrecherche am Beispiel einer LAssi-Modellschulklasse der Gesamtschule Hamburg-Harburg betrachtet. Auf Basis der Ergebnisse dieser Analyse werden anschließend Anforderungen an das zu entwickelnde System definiert. Außerdem wird ein Modell für einen software-gestützten Rechercheprozess eines Schülers entwickelt. Anhand dieses Modells wird ein Ansatz zur Erfüllung der definierten Anforderungen und ein Benutzungsmodell einer diesen Ansatz umsetzenden Software hergeleitet. Ein Fallbeispiel veranschaulicht den gewählten Ansatz. Im Anschluss daran werden der Entwurf und die Implementierung dieses Systems beschrieben. Abschließend wird mit Hilfe vorab vorgestellter Kennzahlen und zusätzlich anhand einer Befragung von Schülern der Modellklasse das Potential des gewählten Ansatzes bewertet. Nach der Darstellung von Anregungen für weitergehende Entwicklungsansätze wird abschließend eine Zusammenfassung und ein Ausblick gegeben.

3 Analyse der Rechercheunterstützung von Schülern

Dieses Kapitel analysiert die Situation von Schülern bei ihren Recherchen im World Wide Web (WWW). Dazu werden zuerst die Ergebnisse von freien Schülerinterviews vorgestellt. Das anschließende Kapitel untersucht die Ansätze von gängigen Suchmaschinen zur Lösung der Schülerprobleme. Abschließend werden diese Lösungsansätze kurz diskutiert.

3.1 Schülerprobleme bei Recherchen im WWW

Zur Analyse der aktuellen Recherchesituation von Schülern sind in einer achten LAssi-Modellschulklasse der freie Interviews sowohl im Klassenverband, als auch in Kleingruppen von drei bis vier Schülern durchgeführt worden. Die Interviews haben ergeben, dass die Schüler zur Informationsbeschaffung für die Bearbeitung von Aufgabenstellungen fast ausschließlich das World Wide Web (WWW) durchsuchen. In seltenen Fällen wird auch die Schulbibliothek konsultiert. 19 von 22 Schülern gaben an, pro Tag fünf oder mehr Suchanfragen an eine WWW-Suchmaschine abzusenden. Hierfür nutzen sie hauptsächlich die Suchmaschine Google, in wenigen Fällen das alternative Produkt Yahoo.

Die Schüler begründen ihre Präferenzen für die Recherche im WWW mit den vielfältigen durchsuchbaren Formaten. Sie können mit Google und Yahoo sowohl auf Webseiten, als auch in Zeitungsartikeln, Bildern, Videos oder Landkarten recherchieren. Außerdem gilt nach Auffassung der Schüler vor allem für Google, dass eine “gute” Anfrage in der Regel auch gut brauchbare Ergebnisse liefert. Die Schüler haben die Beobachtung gemacht, dass sich eine “gute” Anfrage meistens durch mehrere enthaltene Suchbegriffe auszeichnet. Zusätzlich ergänzten sie, dass die Auswahl der Suchbegriffe eine wichtige Rolle spiele.

Die Gespräche ergaben aber auch, dass für die Befragten regelmäßig ein hoher Zeitaufwand nötig ist, um alle gewünschten Informationen zu finden. Im Prinzip gibt es für die Schüler bei Recherchen im WWW zwei Hürden :

- Schwierigkeiten bei der Formulierung von Suchanfragen

  Alle Schüler erklärten, dass es ihnen sehr schwer falle und viel Übung erfordere "gute" Suchbegriffe zur Formulierung von Suchanfragen zu finden. Desweiteren ist nur vier Schülern bekannt gewesen, dass zur Formulierung von Suchanfragen die aussagenlogischen Operatoren UND, ODER und NICHT verwendet werden. Der Operator NICHT, mit welchem z.B. unbrauchbare Treffer aus den Ergebnissen gefiltert werden können, war gänzlich unbekannt. Ebenso neu für die Schüler ist die Möglichkeit zur Eingabe einer Wortfolge innerhalb von Anführungszeichen gewesen. Eine solche Suchanfrage liefert nur Treffer, in denen die Wortfolge exakt so wie in der Suchanfrage enthalten ist.

- Zeitintensive Auswertung von umfangreichen Ergebnismengen

  Die Schüler beklagten, dass ihre Suchanfragen in der Regel sehr umfangreiche Ergebnismengen von mehreren tausend Treffern lieferten und allein die Sichtung der rund fünzig ersten Treffer viel Zeit beanspruche. Die häufig von Suchmaschinen verwendete Listendarstellung der Ergebnisse gibt den Schülern deren sequentielle Bearbeitung vor. Außerdem bietet sich den Schülern ohne das Stellen einer neuen Anfrage keine Möglichkeit die Trefferanzahl, z.B. durch Markierung unbrauchbarer Treffer, zu reduzieren. Desweiteren enthalten Suchanfragen zu derselben Aufgabenstellung meistens große Schnittmengen der Treffer. Daher vergeuden die Schüler häufig viel Zeit mit der Sichtung bereits bekannter Treffer[1].

3.2 Lösungsansätze von Suchmaschinen

Es existieren verschiedene Ansätze, um die Nutzung von Suchmaschinen zu vereinfachen. Dieses Kapitel stellt diejenigen vor, welche insbesondere auf die beschriebenen Probleme der Schüler eingehen. Die Ansätze sind entweder bereits in einer Suchmaschine oder eine öffentlich zugänglichen Testversion implementiert.

[1] Anmerkung: An dieser Stelle hilft die konfigurierbare optische Hervorhebung von Links zu bereits besuchten (und im Browser-Cache zwischengespeicherten) Webseiten durch den Browser wenig, weil nicht erkennbar ist, in Zusammenhang mit welcher Aufgabe eine Webseite besucht worden ist. Es ist kein seltener Fall, dass eine Webseite zu einer Aufgabe absolut unpassend ist, aber essenzielle Informationen für die erfolgreiche Bearbeitung einer anderen Aufgabenstellung enthält.

3.2.1 Vorschlagen alternativer Anfragen

Ein Kernproblem der Schüler ist die Formulierung von “guten” Anfragen. Häufig werden Suchbegriffe in falscher Schreibweise eingegeben. Die Web-Suchmaschine Google versucht dieses Problem durch eine leistungsfähige Rechtschreibprüfung zu kompensieren, welche den Benutzer auf den möglichen Tippfehler aufmerksam macht und zusätzlich einen Verbesserungsvorschlag anbietet. Der Einfachheit halber ist dieser Vorschlag direkt in der Anfragesyntax als Web-Link formuliert worden, sodass der Benutzer ihn einfach per Mausklick in seine Anfrage übernehmen kann (siehe Abbildung 3.1). Das System schlägt nach Formulierung der Anfrage “Hmbur” korrekt den Begriff “Hamburg” vor. Es wird aber nur ein einziger Vorschlag unterbreitet.

Abbildung 3.1: Rechtschreibprüfung der Suchmaschine Google

Ein weiterer Ansatz zur Erleichterung der Formulierung von Suchanfragen ist das Vorschlagen lexikographisch ähnlicher Begriffe während der Eingabe der Suchanfrage. Die Suchmaschine Google hat diesen Ansatz in Form der aktuell noch in einer Testphase befindlichen Software Google Suggest implementiert[2]. Die Abbildung 3.2 zeigt Google Suggest während der Eingabe der Suchanfrage “Hamburg”. Zusätzlich zu den vorgeschlagenen Suchanfragen gibt Google Suggest die erwartete Anzahl der Treffer zu diesen Suchanfragen aus.

Abbildung 3.2: Vervollständigung einer Suchanfrage durch Google Suggest

Ein interessanter Ansatz zur Verbesserung der Trefferausbeute des Benutzers ist das Vorschlagen vollständig neuer Suchbegriffe. Inhaltliche Zusammenhänge von Suchbegriffen können

[2] Google Suggest ist erreichbar unter http://www.google.com/webhp?complete=1&hl=en

z.B. auf Basis statistischer Korrellationsanalysen in einer Trainingsmenge von Dokumenten, einem sog. Korpus, aufgedeckt werden. Eine ausgereifte Implementierung dieses Ansatzes stellt die Suchmaschine Web.de SmartSearch dar, deren Suchbegriffsvorschläge für die Anfrage “Hamburg” auszugsweise in der Abbildung 3.3 dargestellt sind [3].

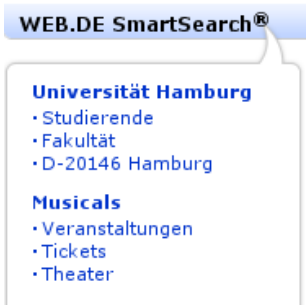

Abbildung 3.3: Vorschläge für alternative Suchbegriffe von Web.de SmartSearch

3.2.2 Benutzereinfluss auf die Trefferanordnung

Das zweite Kernproblem der Schüler besteht in der zeitlich aufwendigen Auswertung der Trefferliste. Die Position eines Treffers in dieser Liste wird als Rang bezeichnet und repräsentiert die Relevanz des Treffers zur Suchanfrage. Ein Kernproblem der Suchmaschinen ist, neben dem “Entdecken” eines Treffers, die Bestimmung dessen Ranges in der Ergebnisliste.

Die Suchmaschine Google zählt die Anzahl der Referenzen aller anderen indexierten Dokumente auf ein bestimmtes Dokument (z.B. in Form von Links in HTML-Dokumenten oder Zitaten in wissenschaftlichen Ausarbeitungen). Je höher die Anzahl der Referenzen auf dieses bestimmte Dokument ist, desto höher ist sein Rang in einer Trefferliste von Google[4].

Die Suchmaschine Yahoo hat einen interessanten Filteransatz in Form des Produktes MindSet entwickelt, welches sich zur Zeit in einer Testphase befindet. MindSet gibt dem Benutzer die Möglichkeit den groben Hintergrund seiner Suchanfrage einzugeben und damit die Ränge der Treffer zu beeinflussen[5].

Die Benutzerschnittstelle von Mindset entspricht der einer klassischen Suchmaschine. Erst nach der Formulierung einer Anfrage und der Präsentation der Treffer in Listenform erlaubt MindSet die Beeinflussung der Trefferränge durch einen Schieberegler mit zwei Extrempolen. Die linke Ausrichtung des Reglers repräsentiert die Benutzerpräferenz für Einkaufsangebote (“Shopping”) und die rechte Ausrichtung für inhaltlich gehaltvolle Dokumente (“Re-

[3] Web.de SmartSearch ist erreichbar unter http://suche.web.de/search/webhp/
[4] vgl. [2], Seite 4 ff.
[5] Yahoo Mindset ist erreichbar unter http://mindset.research.yahoo.com

searching"). Diese Pole sind fest von MindSet definiert und unabhängig von der eingegebenen Suchanfrage. Die im Index von MindSet erfassten Dokumente sind ebenfalls fest einem der beiden Extrempole mit einem bestimmten Polabstand zugeordnet. Wie Yahoo diese Polabstände genau ermittelt, ist nicht dokumentiert. Die angezeigten Treffer werden nach Relevanz zur eingegebenen Suchanfrage bewertet und absteigend sortiert angezeigt, wie es für Suchmaschinen üblich ist. Die Ausrichtung des Reglers repräsentiert den gewünschten Polabstand der wahrscheinlich für den Benutzer hochwertigen Treffer. Nach Betätigung des Reglers ändert sich die Anordnung der angezeigten Treffer in Echtzeit. Für die Anfrage "Hamburg" ist das Ergebnis von Yahoo Mindset stark reduziert in der Abbildung 3.4 dargestellt.

shopping
researching
1. (1) Hamburg - Wikipedia, the free encyclopedia
www.hamburg.de ... League and the fact that Hamburg is a City State and one situated on the southern tip of Jutland ...
en.wikipedia.org/wiki/Hamburg

Abbildung 3.4: Benutzeroberfläche von Yahoo Mindset

3.2.3 Benutzereinfluss auf die Trefferdarstellung

Je kürzer eine Trefferliste ist, desto schneller kann ein Schüler diese vollständig überblicken und ihre Einträge sichten. Die Bestimmung eines Trefferranges soll implizit eine Verkürzung der Trefferliste vornehmen, da z.B. die befragten Schüler nach eigenen Angaben maximal die ersten 50 von mehreren tausend Treffern sichten. Sie verlassen sich auf die Relevanzbewertung durch die genutzte Suchmaschine.

Die Meta-Suchmaschine Grokker des Unternehmens Groxis Inc. verfolgt einen anderen Ansatz zur Reduktion der Trefferliste. Eine Meta-Suchmaschine ist eine Suchmaschine, welche nicht nur oder überhaupt nicht in einem eigenen Index nach Treffern zu einer Anfrage sucht, sondern diese Anfrage an andere Suchmaschinen weiterleitet und deren Ergebnisse präsentiert. Grokker berechnet Gruppen von inhaltlich ähnlichen Treffern. Diese Gruppen werden als Cluster bezeichnet und in Abhängigkeit der enthaltenen Treffer benannt. In der Implementierung von Grokker kann ein Treffer in mehreren Clustern enthalten sein. Der Benutzer sichtet nur die Treffer der Cluster, die ihrer Bezeichnung nach relevant zu sein scheinen.

Dazu fordert Grokker, wie auch Google oder Yahoo, vom Benutzer die Eingabe einer Suchanfrage. Anschließend leitet es die eingegebene Suchanfrage an externe Suchmaschinen weiter und sammelt eine bestimmte Maximalanzahl von Treffern[6]. Danach werden diese

[6]Zur Zeit können Treffer der Suchmaschine Yahoo, aus dem Online-Lexikon Wikipedia und der Literatursuche von der Online-Buchhandlung Amazon einbezogen werden.

auf Basis der in Textform enthaltenen Inhalte zu Clustern zusammengefasst. Ist eine bestimmte Anzahl von Treffern pro Cluster überschritten, so berechnet Grokker aus den enthaltenen Treffern neue Sub-Cluster. Dieser Schwellenwert kann durch den Benutzer nicht konfiguriert werden. Nach der Berechnung der Cluster erhält der Benutzer diese wahlweise in grafischer oder textueller Darstellung. Die grafische Darstellung erfolgt in Form von benannten Kreisen innerhalb derer weitere Cluster oder Trefferdokumente grafisch angedeutet werden. Die Abbildung 3.5 zeigt beispielhaft die Grokker-Ergebniscluster zur Suchanfrage “Hamburg”.

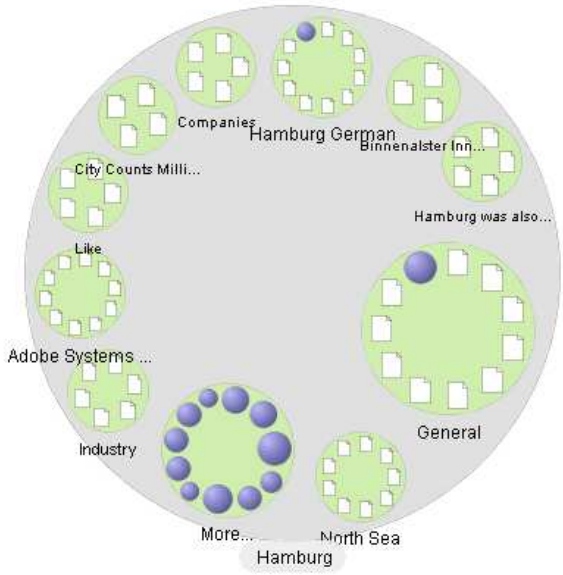

Abbildung 3.5: Grafische Darstellung der Treffercluster von Grokker

Die textuelle Darstellung gibt eine Aufzählung der Clusterbenennungen in Form von Web-Links aus. Nach einem Mausklick auf einen Kreis bzw. einen Link öffnet sich in einem seperaten Bildschirmbereich eine Liste der im selektierten Cluster enthaltenen Treffer. Die Darstellung dieser Liste ähnelt den Ergebnislisten von Google oder Yahoo.

3.3 Fazit

Die befragten Schüler haben gegenüber Suchmaschinen den Anspruch ihre gewünschten Informationen mit minimalem Zeiteinsatz zu finden. Ihre Kernprobleme bestehen im Finden und Formulieren von Suchanfragen, sowie im hohen Zeiteinsatz bei der Trefferauswertung. Ein für die Schüler nutzbarer und bereits bestehender Ansatz zur Lösung beider Kernprobleme ist gar nicht gefunden worden.

Eine Rechtschreibprüfung für eingegebene Suchbegriffe ist ohne Zweifel sinnvoll, um für

“gut” ausgewählte Suchbegriffe die bestmögliche Trefferausbeute zu erzielen. Allerdings stellt bereits die Auswahl von “guten” Suchbegriffen eine Hürde für die Schüler dar. Die lexikografische Vervollständigung der Suchbegriffe bei deren Eingabe durch Google Suggest ist hier ebenfalls unzureichend, da die Vorschläge eben nur lexikografisch, aber nicht inhaltlich ähnlich sind.

Der im Bezug auf die Unterstützung zur Wahl von Suchbegriffen fortgeschrittenste Ansatz ist die Web.de SmartSearch. Dieses Programm schlägt vollkommen neue Suchbegriffe vor. So ist es dem Schüler möglich, seine Suchanfragen durch Ergänzung eines oder mehrerer zusätzlicher Suchbegriffe über den gesamten Suchprozess hinweg zu verbessern, an die er selbst gar nicht gedacht hat. Google Suggest hingegen leitet nach einmaliger Unterstützung zur normalen Google-Suche weiter. Leider ist es mit der Web.de SmartSearch nur möglich, Positivkriterien in Form von zusätzlichen Suchbegriffen anzugeben. Ein Schüler kann dieses System nicht anhand eines unbrauchbaren Treffers anweisen, alle restlichen derartigen Treffer aus der Ergebnismenge zu entfernen. Außerdem entbindet keiner der vorgestellten Ansätze den Schüler von der Notwendigkeit mindestens eine initiale Suchanfrage zu formulieren.

Ein besonders interessanter Ansatz im Hinblick auf die Schülerunterstützung bei der Formulierung der Suchanfragen ist das MindSet-System der Suchmaschine Yahoo. Der Vorteil des Systems ist, dass ein Schüler ohne Änderung seiner Suchanfrage die Sortierung der Treffer ändern kann um die für ihn, vor dem Hintergrund seiner Aufgabenstellung, relevanten Treffer zuerst angezeigt zu bekommen. Leider ist die Einteilung aller Treffer beliebiger Suchanfragen in die zwei Kategorien “Shopping” und “Research” statisch vorgeben und kann, abhängig von der gestellten Suchanfrage, nicht geändert werden.

Die Nutzung von Clustern zur Strukturierung der angezeigten Treffermenge, wie Grokker sie anbietet, könnte den benötigten Zeitaufand der Schüler zur Sichtung der Treffer deutlich senken. Die Schüler müssten nicht Treffer in Clustern sichten, deren Bezeichnung auf einen fehlenden Bezug zur Aufgabenstellung hindeutet. Außerdem können sie davon ausgehen, dass die restlichen Treffer eines Clusters mit wenigen gesichteten und unpassenden Treffern sehr wahrscheinlich ebenfalls unbrauchbar sein werden. Auf diese Weise sind die Schüler in der Lage derlei Cluster zu verwerfen, ohne alle Treffer in diesen Clustern gesichtet zu haben.

4 Skizzierung des zu entwickelnden Systems

4.1 Anforderungsdefinition

Auf Basis der geführten Schüler-Interviews ist der folgende Katalog funktionaler, allgemeiner und technischer Anforderungen an das zu entwickelnde System zusammengestellt worden:

Funktionale Anforderungen:

- Es muss einem Schüler mit dem System möglich sein, ohne Kenntnis und Nutzung der aussagenlogischen Operatoren, präzise Treffer zu einer Aufgabenstellung zu recherchieren.
- Die Menge der recherchierten Treffer muss für einen Schüler mit minimalen Zeitaufwand vollständig zu überblicken sein, sodass dieser irrelevante Treffer schnell identifizieren kann und diese nicht komplett sichten muss.
- Minimale Trefferredundanz: Ein recherchierter und vom Schüler ausgewerteter Treffer soll pro Aufgabenstellung nur ein einziges Mal als neuer Treffer angezeigt werden.
- Die Treffermenge muss durch die Entfernung irrelevanter Treffer reduzierbar sein.
- Das System muss den Schüler bei seiner aktuellen Aufgabenstellung entsprechend seines Kenntnisstandes unterstützen.

Allgemeine Anforderungen:

- Das System muss von einem, in der Bedienung von zeitgemäßen Personal Computern erfahrenen, Schüler der siebten Klassenstufe installierbar und ohne umfangreiche Einarbeitung bedienbar sein.
- Die Unterstützung des Systems muss auf jede Suchmaschine ausdehnbar sein. Daher sollte prinzipiell jede Suchmaschine in das System integriert werden können.

Technische Anforderungen:

- Der Betrieb des Systems darf aus Kostengründen keine Anpassungen der IT-Infrastruktur einer Schule, in Form des Erwerbs zusätzlicher Hard- und Software, erfordern.
- Das System muss in jeder technischen Umgebung lauffähig sein, in der der Betrieb der LAssi-Plattform möglich ist.

4.2 Entwicklung eines Lösungsansatzes

4.2.1 Abbildung einer Aufgabenstellung in der LAssi-Plattform

Die Unterstützung der Schüler bei Recherchen soll bezogen auf deren aktuelle Aufgabe erfolgen. Daher muss geklärt werden, wie Aufgabenstellungen in der LAssi-Plattform abgebildet werden. Die LAssi-Plattform ist im Einleitungskapitel als individuelle Lernumgebung vorgestellt worden. Konkret heißt das, dass jeder Schüler sein eigenes LAssi-System mitsamt einer individuellen Zusammenstellung von digitalen Lernwerkzeugen besitzt, welches auf einem USB-Stick abgespeichert ist. Die Lernwerkzeuge werden auf einem speziellen Server, z.B. im Netzwerk der Schule oder im Internet, bereitgestellt und können von den Schülern nach einem Plugin-Konzept in ihren “LAssi” “hinzugesteckt” bzw. “entfernt” werden. Mit diesem Werkzeugkasten bearbeitet der Schüler Aufgabenstellungen aus dem Unterricht. Erzeugnisse dieser verschiedenen Lernwerkzeuge werden im LAssi-System als Materialien bezeichnet, z.B. ein Karteikartenmaterial als Erzeugnis eines Karteikarteneditors.

Um die werkzeugübergreifende Arbeit mit Materialien zu ermöglichen, verfügt das LAssi-System über ein eigenes Desktop-Konzept. Hinter diesem Konzept steckt die Metapher eines Schreibtisches, auf dem Materialien, die zur Bearbeitung einer Aufgabenstellung relevant sind, abgelegt werden können. Anders als in gängigen Desktop-Betriebssystemen, wie Microsoft Windows oder Apple MacOS, existiert in einem LAssi-System nicht zwangsläufig ein einziger Desktop, sondern nach dem LAssi-Benutzungsmodell genau einer für jede Aufgabenstellung, die vom Schüler bearbeitet wird. Während der gesamten Laufzeit des LAssi-Systems ist immer ein einziger Desktop geöffnet. Beim Anlegen eines neuen Desktops muss der Schüler eine Bezeichnung für den neuen Desktop angeben.

Die LAssi-Desktops dienen zum einen als Austauschplattform für Materialien zwischen einzelnen Lernwerkzeugen und zum anderen als Räume für elementare Strukturierungen, z.B. die Bildung von Gruppen zusammengehöriger Materialien in einem Stapel. Einige Materialien, wie z.B. Karteikarten, können Referenzen auf andere Materialien oder auf Dateien in einem lokalen (z.B. ein Microsoft Word-Dokument) oder entfernten Dateisystem (z.B. eine

Webseite) besitzen. Daher können sowohl alle LAssi-Materialien als auch Erzeugnisse externer Programme (z.B. Webseiten oder Word-Dokumente), die zu einer Aufgabenstellung gehören, über den zur Aufgabe gehörenden LAssi-Desktop erreicht werden.

In der LAssi-Plattform wird folglich nicht nur die Aufgabenstellung an sich, sondern auch ihr Bearbeitungszustand durch einen LAssi-Desktop abgebildet. Die Lernmaterialien und weiterführenden Quellen, sowie die Zwischen- und Endprodukte der Schüler können eindeutig dieser Aufgabenstellung zuordnet werden. Daher erscheint es sinnvoll, den aktuell geöffneten LAssi-Desktop des Schülers als dessen Suchprofil zu nutzen.

4.2.2 Herleitung des Lösungsansatzes

Die geführten Interviews haben ergeben, dass für die Schüler eine Herausforderung beim Durchsuchen des WWW im Finden von Suchbegriffen und im Formulieren von Suchanfragen besteht. Die bestmögliche Entlastung der Schüler besteht in der vollständigen Automatisierung der Anfragengenerierung unter Erzielung von, für einen Schüler, gut brauchbarer Treffer. Dies könnte beispielsweise durch einen, in die LAssi-Plattform eingebetteten, Assistenten erledigt werden, welcher Suchbegriffe vorschlägt und aus diesen, in Syntax einer ausgewählten Suchmaschine, auch Anfragen formuliert. Um Suchbegriffe durch einen solchen Assistenten finden zu lassen, wird die Erfüllung zweier Voraussetzungen benötigt:

- Existenz einer Menge von Kandidaten für Suchbegriffe
- Ein Auswahlverfahren für Suchbegriffe aus der Kandidatenmenge

Um eine Suchanfrage zu berechnen, die dem Schüler Treffer zu seiner aktuellen Aufgabenstellung liefert, müssen die Suchbegriffkandidaten in inhaltlichem Zusammenhang zur Aufgabenstellung des Schülers stehen. Lernmaterialien und Quellen, welche der Schüler in Zusammenhang mit seiner Aufgabenstellung bringt, bieten sich daher als Quelle von Kandidaten für Suchbegriffe an und finden sich auf dem LAssi-Desktop zur Aufgabe. Die Berechnung von konkreten Kandidaten aus dem Inhalt des LAssi-Desktops lässt sich auf das Problem der Berechnung von charakteristischen Begriffen für eine Menge von Texten zurückführen. Dieses Problem kann mit statistischen Verfahren aus der Disziplin Information Retrieval auf Basis verschiedener Worthäufigkeitsansätze gelöst werden.

Die Verwendung von Suchbegriffen aus dem Arbeitskontext des Schülers bedeutet allerdings nicht zwingend, dass alle erzielten Treffer für den Schüler bei der Bearbeitung seiner Aufgabenstellung nützlich sind. Außerdem soll der Schüler während des gesamten Bearbeitungsprozesses seiner Aufgabenstellung unterstützt werden und es ist zu erwarten, dass dazu mehrere Recherchen in unterschiedliche inhaltliche Richtungen notwendig sind. Daher muss

der Schüler die Möglichkeit haben “seinen” Assistenten bei der Berechnung von Suchbegriffen zu steuern.

Ein sehr anschaulicher Ansatz wäre ein Bewertungssystem für Treffer von automatisch generierten Suchanfragen. In so einem System können Treffer vom Schüler z.B. als “brauchbar” oder “unbrauchbar” markiert werden. Die Konsequenz der Bewertungen wäre, dass der Assistent mit seinen zukünftigen Vorschlägen für Anfragen das Erzielen von “brauchbaren” Treffern begünstigt und von “unbrauchbaren” vermeidet. Hierfür könnte der Assistent aus den “brauchbaren” Treffern, ebenso wie aus dem Inhalt des aktuellen LAssi-Desktops, Kandidaten für Suchbegriffe extrahieren. Aus “unbrauchbaren” Treffern wären auf dieselbe Weise Filterbegriffe extrahierbar. Das sind Begriffe die, mit dem logischen Operator NICHT versehen, an die Suchanfrage angehängt werden. Hiermit wird erreicht, dass nur Treffer erzielt werden, welche frei von Filterbegriffen sind. Ein Modell für den Rechercheprozess mit Unterstützung eines solchen Assistenten ist in der Abbildung 4.1 dargestellt.

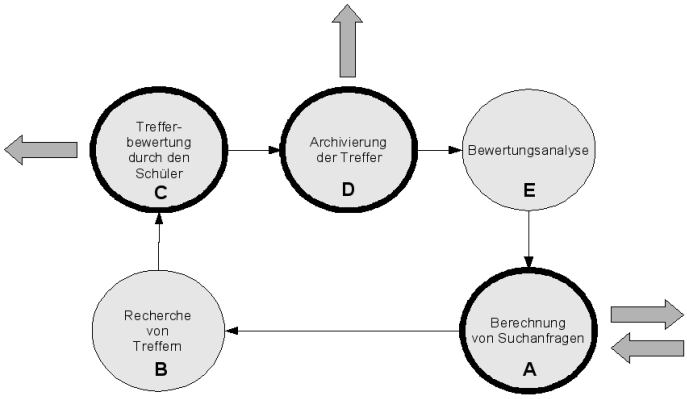

Abbildung 4.1: Modell eines unterstützten Rechercheprozesses für Schüler

Die Ein- und Austrittsschritte des Prozesses sind fett umrandet. Der Prozess beginnt mit der Berechnung der ersten Suchanfrage (Schritt A), deren Suchbegriffe aus Materialien vom aktuellen LAssi-Desktop stammen. Sind keine Materialien vorhanden, so wird der Prozess auch nicht fortgesetzt. Der Schüler übernimmt anschließend die vom Assistenten ausgegebene Suchanfrage in das Anfrageformular einer Suchmaschine und startet deren Suchprozess (Schritt B). Er sichtet und bewertet die erzielten Treffer (Schritt C). An dieser Stelle muss der Schüler entscheiden, ob er einen weiteren Vorschlag für eine Anfrage durch den Assistenten einholen oder den Recherchevorgang abbrechen möchte, um die erzielten Treffer in seine Lösung der Aufgabenstellung einzupflegen. Entscheidet sich der Schüler für einen weiteren Vorschlag, so muss er dem Assistenten die Treffer und deren Bewertungen bereitstellen. Dazu könnten die “brauchbaren” und “unbrauchbaren” Treffer z.B. auf dem LAssi-USB-Stick, je

nach Bewertung, in eines von zwei Verzeichnissen gespeichert werden. Diese Archivierung der Treffer wird durch Schritt D abgedeckt. Die Inhalte dieser zwei Verzeichnisse werden vom Assistenten in Schritt E ausgelesen und analysiert, um Kandidaten für die Berechnung von Suchanfragen in Schritt A zu erzeugen. Mit erneutem Beginn des Schrittes A schließt sich der Kreislauf der Prozesses. Da der Assistent die Inhalte des aktuellen LAssi-Desktops bei der Auswahl der Suchbegriffkandidaten berücksichtigt, kann der Assistent die Qualität seiner Vorschläge, auf Basis von Dokumenten oder LAssi-Materialien, verbessern, die er nicht selbst gefunden hat. Auf diese Weise basieren die Vorschläge für Suchanfragen des Assistenten immer sowohl auf dem aktuellen Bearbeitungszustand der Aufgabenstellung, als auch auf den bisher gefundenen und vom Schüler bewerteten Treffern.

Der beschriebene Assistent entbindet die Schüler vollständig vom Finden der Suchbegriffe und der Formulierung der Suchanfragen. Außerdem ist er für jede Suchmaschine einsetzbar, die boole'sche Anfragen entgegennimmt, da zur Formulierung der Anfrage in korrekter Syntax lediglich die Hinterlegung der Operatornotationen der jeweiligen Suchmaschine notwendig ist (beispielsweise "+", "AND" oder "UND" für das logische UND). Allerdings löst er nicht das zweite Kernproblem der befragten Schüler, nämlich ihren hohen Zeiteinsatz bei der Auswertung der erzielten Treffer. Durch die vorgeschlagenen Suchanfragen ist zwar eine Reduktion der gesamten Treffermenge möglich, aber eine Zeitersparnis durch Verbesserung der Darstellung der erzielten Treffer, durch Zusammenfassung ähnlicher Treffer in Clustern oder Ausblendung redundanter Treffer bei mehreren Suchanfragen, wird nicht erreicht. Die Schüler müssen hingegen zusätzlich Zeit für die Übertragung der vorgeschlagenenen Suchanfrage in die Eingabemaske der Suchmaschine und Bereitstellung der erzielten Treffer für den Assistenten investieren. Daher genügt der Assistent den formulierten Anforderungen in der beschriebenen Form noch nicht.

Der beschriebene zusätzliche Zeiteinsatz des Schülers durch Nutzung des Assistenten entfällt, wenn der Assistent selbst mit den vom Schüler genutzten Suchmaschinen kommuniziert. Das bedeutet, er muss seine Anfragen an die Suchmaschine stellen und deren Treffer empfangen. Diese Lösung hat zwar den Nachteil, dass für jede Suchmaschine ein Adapter in den Assistenten integriert werden muss, welcher die Kommunikation beider Systeme (Assistent und Suchmaschine) abwickelt. Aber da die Menge der von den Schülern genutzten Suchmaschinen endlich und sehr klein ist, kann dieser Nachteil vernachlässigt werden. Durch die Suchmaschinenintegration könnte zusätzlich generisch für alle Treffer aller vom Schüler genutzten Suchmaschinen eine einzige Trefferdarstellung, z.B. in der beschriebenen Clusterform, angeboten werden. Außerdem ist es dem Assistenten möglich, durch verschiedene Suchanfragen doppelt gelieferte Treffer auszusortieren und vom Schüler bewertete Treffer auszublenden, sodass sich dieser auf die Sichtung der unbekannten Treffer konzentrieren kann.

Der Assistent könnte dem Schüler außerdem viel Zeit ersparen, wenn er diesen an bereits erworbenes Wissen erinnert. Dazu könnte der Assistent auf die LAssi-Desktops, also die Aufgabenstellungen, des Schülers zurückgreifen. Der Schüler könnte dieses, von ihm selbst bereits aufbereitete und verstandene, Wissen nutzen und eine weitere Recherche wäre gar nicht erforderlich. Außerdem ist dieses Wissen für den Schüler von einer wesentlich höheren Qualität als z.B. ein Treffer von Google, da dieses Wissen von ihm selbst aufbereitet und in einen bekannten Gesamtkontext (die andere Aufgabenstellung) eingeordnet worden ist.

Zur Umsetzung dieses Ansatzes müsste die LAssi-Volltextsuche über einen Adapter in den Assistenten integriert werden. Auf diese Weise könnte der Assistent Treffer auf anderen LAssi-Desktops recherchieren und dem Schüler diese zusammen mit den Treffern von Google und Yahoo in Clustern darstellen. Diese Darstellung hat für den Schüler den Vorteil, dass sein eigenes relevantes Wissen in Beziehung zu dem externen, im WWW verfügbaren, Wissen gesetzt wird. So kann er das externe Wissen schneller klassifizieren und auch leichter verstehen. Da in der Clusterdarstellung die Treffer aller integrierten Suchdienste gemeinsam in Cluster einsortiert werden, ist diese Ansicht zur schnellen Erkennung aller Treffer der LAssi-Suche ungeeignet. Hierbei könnte eine Herkunftsansicht nützlich sein, die Treffer nicht in Abhängigkeit zu deren Inhalt, sondern zur Suchmaschine clustert, die den Treffer erzielt hat. So ist es dem Schüler möglich, nur Treffer bestimmter Suchmaschinen anzuzeigen.

4.2.3 Entwicklung eines Benutzungsmodells

Da der Assistent die Berechnung von Suchanfragen, ihre Übermittlung an Suchmaschinen und die Entgegennahme deren Treffer vollständig automatisiert, bestehen die Interaktionsmöglichkeiten des Schülers mit dem Assistenten hauptsächlich in dessen Steuerung:

- Der Schüler kann die Unterstützung durch den Assistenten ein- und abschalten. Sobald der Assistent Treffer recherchiert hat, benachrichtigt er den Schüler.
- Der Schüler kann die inhaltliche Richtung zukünftiger Recherchen des Assistenten steuern. Dazu bewertet er erzielte Treffer.

Der Assistent besitzt eine gewisse Autonomie, da der Schüler ihn nur starten und beenden kann. Der Assistent recherchiert im Hintergrund Treffer für den Schüler und informiert ihn, wenn eine bestimmte Anzahl Treffer erzielt oder alle möglichen Suchanfragen gestellt worden sind. Dies hat den Vorteil, dass der Schüler durch den Assistenten nicht in seiner Arbeit beeinträchtigt wird. Die Recherchen des Assistenten sollen in bestimmten zeitlichen Abständen erfolgen, welche durch den Schüler zu spezifizieren sind.

Durch die Bewertungen des Schülers erhalten die Treffer einen der folgenden qualitativen Zustände:

- Zu berücksichtigender, hochwertiger Treffer

  Zu berücksichtigende, hochwertige Treffer helfen dem Schüler bei der Lösung eines bestimmten Teilbereiches seiner Aufgabenstellung. Der Assistent berücksichtigt Treffer mit diesem Zustand bei der Auswahl zukünftiger Suchbegriffe. Ein hochwertiger Treffer ist nur solange zu berücksichtigen, wie er zur inhaltlichen Rechercherichtung des Schülers gehört. Sobald der Schüler eine andere Rechercherichtung einschlagen möchte, muss der Assistent sehr wahrscheinlich andere Suchbegriffe wählen. Daher muss der Schüler ihm die bisher zu berücksichtigenden, hochwertigen Treffer als Quelle für Suchbegriffkandidaten zu entziehen und in eine Ablage verschieben (hochwertiger Treffer).

- Hochwertiger Treffer

  Ein inhaltlich hochwertiger Treffer hilft dem Schüler ebenfalls bei der Lösung seiner Aufgabenstellung. Allerdings berücksichtigt der Assistent die Inhalte hochwertiger Treffer nicht bei der Wahl zukünftiger Suchbegriffe. Dieser Zustand dient daher als Ablagezustand und ist sinnvoll, um z.B. Quellen zu sammeln, welche zur Lösung der Aufgabenstellung beigetragen haben.

- Neutraler Treffer

  Jeder Treffer ist standardmäßig neutral und muss vom Schüler gesichtet werden, um anschließend eine Bewertung zu erhalten. Neutrale Treffer werden vom Assistenten nicht berücksichtigt.

- Geringwertiger Treffer

  Ein geringwertiger Treffer ist für den Schüler in keiner Weise zur Lösung seiner Aufgabenstellung nützlich. Daher nutzt der Assistent alle geringwertigen Treffer als Quelle für Filterbegriffe, um in zukünftigen Recherchen Treffer dieser Art zu vermeiden. Anders als zu berücksichtigende, hochwertige Treffer kann ein geringwertiger Treffer niemals in eine isolierte Ablage verschoben werden, sondern ist immer für die Berechnung von Filterbegriffen zu berücksichtigen (bei aktueller Aufgabenstellung).

Alle Treffer mit demselben Zustand werden in eine gemeinsame Zustandsmenge gelegt. Der Schüler kann sich die Zustandsmengen jeweils als eine mit Treffern gefüllte Schale vorstellen. Es gibt daher eine Schale mit Treffern zum Fokus der Aufgabenstellung (zu berücksichtigende, hochwertige Treffer), eine Ablageschale für nützliche Treffer (hochwertige Treffer), eine

Sammelschale, in die alle neuen Treffer vom Assistenten gelegt werden, (neutrale Treffer) und eine Abfallschale (geringwertige Treffer). Die Inhalte der Schalen beziehen sich immer auf die bearbeitete Aufgabenstellung, also auf den aktuellen LAssi-Desktop. Folglich gehören zu jedem LAssi-Desktop eigene Schaleninhalte. Die Schaleninhalte können nur vom Assistenten bereitgestellt werden. Es ist also nicht möglich, dass der Schüler z.B. Bookmarks aus seinem Browser in die Schalen einfügt. Wenn der Schüler dem Assistenten ein bestimmtes externes Dokument, z.B. eine Webseite, zugänglich machen will, so muss er dieses Dokument als Anhang an ein LAssi-Material auf seinem LAssi-Desktop hängen. Es ist nicht vorgesehen, dass der Schüler vom aktuellen LAssi-Desktop auf die Schaleninhalte anderer Desktops zugreift. Schließlich werden die Inhalte der nützlichen recherchierten Treffer vom Schüler in seine LAssi-Materialien eingearbeitet. Die relevanten LAssi-Materialien anderer Desktops werden vom Assistenten wiederum über die integrierte LAssi-Suchmaschine gefunden.

Alle Treffer sollen vom Schüler direkt in einem geeigneten Viewer oder Editor geöffnet werden können, z.B. von einem Webbrowser im Falle einer Webseite. Außerdem muss es möglich sein, dass der Schüler einen Treffer auch endgültig aus dessen Schale löschen kann. Ein gelöschter Treffer darf vom Assistenten nicht erneut gefunden werden. Damit der Schüler bei umfangreichen Schalen nicht die Übersicht über bereits gesichtete und ungesichtete Treffer verliert, soll ein Treffer als “gelesen” oder “ungelesen” angezeigt werden (vergleichbar mit gelesenen und ungelesenen Mails in E-Mail-Clients).

Angenommen der Schüler hat die Unterstützung seines Assistenten angefordert und ignoriert aber dessen Hinweise auf recherchierte Treffer, so fügt der Assistent immer mehr Treffer in die Sammelschale ein. Dies führt zu unnötig hohem Speicherverbrauch, da die Schalen auf dem LAssi-USB-Stick persistiert werden. Zudem ist eine umfangreiche Sammelschale nur zeitintensiv für den Schüler auszuwerten, vor allem weil er Zeit in die Sichtung von Treffern investieren muss, die sich auf einen weit zurückliegenden Bearbeitungsstand der Aufgabe beziehen. Daher erscheint es sinnvoll, die maximale Anzahl von Treffern in einer Schale zu beschränken. Aber was soll passieren, wenn eine Schale voll ist? Es wäre möglich, den Assistenten in diesem Fall zu beenden und erst dann wieder zu starten, wenn der Schüler die Sammelschale geleert hat. Diese Variante hat den Nachteil, dass das Eingreifen des Schülers verlangt wird. Sinnvoller ist es, die ungelesenen Treffer aus der Sammelschale durch die neu recherchierten Treffer zu ersetzen. Erst wenn alle Treffer der Sammelschale als gelesen markiert sind, wird der Recherchevorgang durch den Assistenten abgebrochen.

Grundsätzlich stellt sich die Frage, welchen Mehrwert persistente Trefferschalen für den Schüler haben. Während der Bearbeitung einer Aufgabenstellung in mehreren Sitzungen mit dem LAssi-System haben sie sicher einen hohen Mehrwert, da der Assistent die Treffer des Schülers archiviert. Der Schüler kann seine Quellen so komfortabel und unabhängig vom

Computer, an dem er arbeitet, nutzen, denn er trägt die Quellen auf seinem LAssi-USB-Stick mit sich. Sie kosten damit Speicherplatz auf dem Stick. Nach Abschluss einer Aufgabenstellung ist dies aber nicht mehr zwingend erforderlich, denn der Schüler hat dann das relevante Wissen aus den Treffern auf LAssi-Materialien extrahiert und aufbereitet. Es muss also eine Möglichkeit existieren, die persistenten Schaleninhalte vom LAssi-USB-Stick löschen. Die Frage des Nutzens von persistierten Trefferschalen ist damit aber noch nicht abschließend geklärt und bietet Raum für weitere Untersuchungen.

4.3 Veranschaulichung anhand eines Fallbeispiels

In der befragten achten Klasse erhalten die Schüler von ihrem Lehrer im Fach Biologie die Aufgabe, sich mit Lebensräumen und Charakteristika von Garnelen zu beschäftigen. Am Ende der Unterrichtseinheit soll ein Vortrag vor der Klasse gehalten werden. Der Lehrer gibt die Bearbeitung dieser Aufgabe in drei Schritten vor:

1. Beschaffen einer Informationsgrundlage in Zweiergruppen
2. Gemeinsames Festlegen der genauen Vortragsthemen im Klassenverband
3. Ausarbeiten des Vortrages in Zweiergruppen

Die Schüler bilden Zweiergruppen und starten das LAssi-System eines Gruppenmitglieds auf einem Schulrechner. Anschließend wird ein neuer LAssi-Desktop mit der Bezeichnung “Die Garnele” angelegt und der LAssi-Rechercheassistent durch Mausklick auf eine spezielle Schaltfläche aktiviert. Als Suchdienste sind die LAssi-Suche und die Web-Suchmaschinen Google und Yahoo integriert[1].

Da sich noch keine weiteren LAssi-Materialien auf dem Desktop befinden, berechnet der Rechercheassistent seine erste Anfrage aus der Bezeichnung des Desktops. Nachdem er alle Suchmaschinen mit dieser Anfrage befragt und deren Treffer entgegengenommen hat, informiert er die Schüler über seine erzielten Treffer, z.B. mit einer aufklappenden Hinweismeldung. Diese öffnen danach das Ergebnisfenster des Assistenten. Dort wird ihnen standardmäßig die Herkunftsansicht angezeigt, welche als Entwurf in der Abbildung 4.2 dargestellt ist. Die LAssi-Suchmaschine wird in dieser Abbildung nicht angezeigt, weil sie zu Anfragen des Assistenten keinen Treffer liefern konnte.

Im Entwurf werden die Titel ungelesener Treffer fett und die gelesener normal ausgegeben. Der Wikipedia-Artikel zur Garnele ist demnach bereits von einem Schüler geöffnet worden,

[1] Diese Web-Suchmaschinen sind ausgewählt worden, weil sie von den Schülern präferiert werden.

Abbildung 4.2: Entwurf der Herkunftsansicht mit Treffern

die restlichen nicht. Die Schüler können die Treffer mit einem Doppelklick auf deren Titel in einem geeigneten Programm öffnen, z.B. in einem Webbrowser. Ein Rechtsklick auf einen Treffertitel lässt ein Kontextmenü aufklappen, welches Operationen zum Verschieben des selektierten Treffers in andere Schalen oder dessen Löschen anbietet. Über das Verschieben der Treffer in eine bestimmte Schale wird deren Bewertung durch den Schüler abgebildet. Außerdem steht eine Operation zur Markierung ungelesener Treffer als gelesen und umgekehrt bereit. Um den Inhalt einer anderen Schale ansehen zu können, selektiert der Schüler die gewünschte Schale aus der entsprechend beschrifteten Auswahlbox. Die Liste aller Schalen wird nach einem Linksklick auf den rechten Pfeil der Auswahlbox angezeigt.

Um einen inhaltlichen Überblick über die mit Google und Yahoo erzielten Treffer zu erhalten, wechseln die Schüler über ein Menü in die Clusteransicht. Diese ist für dieselben Treffer in der Abbildung 4.4 dargestellt. Die Bedienung dieser Ansicht ist mit der der Herkunftsansicht identisch (Öffnen der Treffer, Verschieben der Treffer in andere Schalen, etc.).

Mit Hilfe der Herkunftsansicht erkennen die Schüler sofort, dass sie zur Garnele direkt noch keine Aufgabenstellung bearbeitet haben. Daher sind sie auf die Informationen der im Web recherchierten Treffer angewiesen. Dank der Clusteransicht erhalten die Schüler einen Überblick über alle recherchierten Treffer und können diese inhaltlich einordnen. Einkaufsmöglichkeiten für Zierfische und anderes Aquaristikzubehör sind für die Aufgabenstellung der Schüler irrelevant, daher brauchen Sie den Treffer des Clusters “Shop” nicht zu sichten. Von den restlichen zehn Treffern behandeln wahrscheinlich sechs das Thema “Garnele”, daher werden sie diese in jedem Fall auswerten müssen. Anhand der Anzeige der Titel lässt sich die Anzahl der zu sichtenden Treffer um einen weiteren senken, denn der Treffer mit dem Titel “Rezepte” wird sich höchstwahrscheinlich mit den kulinarischen Qualitäten bestimmter Garnelenarten beschäftigen. Die vier Treffer zum Thema Garnelenzucht wären dann inter-

Abbildung 4.3: Entwurf der Clusteransicht mit Treffern zur Anfrage “Garnele”

essant, wenn im Schülervortrag auf die wirtschaftliche Nutzung von Gernelenarten oder die Rolle bestimmter Garnelenarten in z.B. der menschlichen Nahrungskette eingegangen werden soll. Je nachdem welche Relevanz die Garnelenzucht für die Schüler hat, könnten sie an dieser Stelle z.B. inhaltliche Aufgaben untereinander verteilen: der eine Schüler wertet die Treffer zur Garnele aus und der andere beschäftigt sich mit der wirtschaftlichen Nutzung der Garnelen.

Bei der Auswertung stellen die Schüler fest, dass Ihnen einige Treffer überhaupt nicht bei der Erstellung ihres Vortrags helfen werden, so z.B. die Einkaufsangebote und die Rezepte. Sie verschieben beide Treffer über das Kontextmenü in die Abfallschale. Nach der Auswertung aller Treffer würden die Schüler gern mehr über das Wattenmeer erfahren, um Hintergrundinformationen zum Lebensraum der Nordseegarnele zu erhalten. Um den Assistenten in diese inhaltiche Richtung zu steuern, verschiebt der Schüler den Treffer “Lebensraum Wattenmeer” über das Kontextmenü in die Schale der zu berücksichtigenden, hochwertigen Treffer. Der Assistent wird dann bei seiner nächsten Recherche u.a. mit Suchbegriffen aus diesem Treffer suchen. Die übrigen Treffer werden über das Kontextmenü in die Ablageschale verschoben, weil ihre Inhalte für den Vortrag nützlich sein könnten. Nach Ablauf der konfigurierten Zeitspanne zwischen zwei Recherchen startet der Assistent den nächsten Recherchedurchgang. Dabei wird neben dem Inhalt des LAssi-Desktops auch der Treffer “Lebensraum Wattenmeer” bei der Auswahl der Suchbegriffe berücksichtigt.

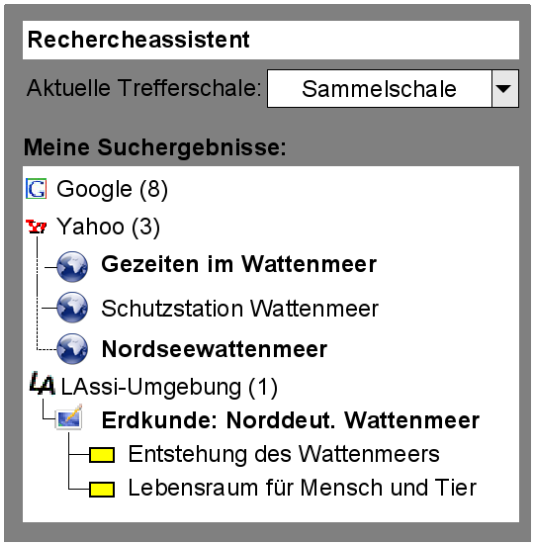

Abbildung 4.4: Entwurf der Herkunftsansicht mit Treffern zum Thema “Wattenmeer”

Die Abbildung 4.4 zeigt die Treffer des zweiten Recherchedurchgangs in der Herkunftsansicht. Hier tritt die Erinnerungsfunktion des Assistenten in Erscheinung, denn der Assistent hat eine Aufgabenstellung aus dem Erdkundeunterricht auf dem LAssi-USB-Stick entdeckt, in welcher sich der Schüler mit dem norddeutschen Wattenmeer auseinander gesetzt hat. Ihm selbst war aufgrund der Fächerbarriere in seinem Kopf gar nicht bewusst, dass er sich bereits Wissen zu diesem Thema aufbereitet und angeeignet hat. Dieses Wissen könnte nun erneut relevant sein und in die aktuelle Aufgabenstellung eingebracht werden.

Die Schüler haben nach Auswertung der erzielten Treffer entweder die Möglichkeit, den Assistenten weiter nach Treffern zum Thema “Wattenmeer” recherchieren, oder eine andere inhaltliche Richtung einschlagen zu lassen. Um die Richtung beizubehalten, müssen sie nichts tun. Zur Änderung der Rechercherichtung ist die Angabe anderer hochwertiger, zu berücksichtigender Treffer notwendig. Dazu wird der Treffer “Lebensraum Wattenmeer” in die Ablageschale verschoben und durch z.B. “Garnele - Wikipedia” und “Hausarbeit: Die Garnele” ersetzt, um mehr über verschiedene Garnelenarten zu erfahren.

4.4 Abgrenzung zu verwandten Systemen

Worin unterscheidet sich der beschriebene Assistent in seiner Funktionalität von anderen Suchmaschinen? Das Benutzungsmodell der gängigen Suchmaschinen ist anfrageorientiert entwickelt worden. Konkret bedeutet das, dass sie vom Schüler eine Anfrage entgegennehmen und relevante Treffer liefern. Der Assistent hingegen unterstützt den gesamten Rechercheprо-

zess eines Schülers: Er berechnet automatisch kontextsensitive Suchbegriffe aus dem Inhalt des aktuellen LAssi-Desktop und der Schale der hochwertigen, zu berücksichtigenden Treffer, formuliert Suchanfragen und präsentiert die erzielten Treffer. Anschließend erwartet er vom Schüler Rückmeldung über sowohl die Qualität seiner Treffer, als auch die inhaltliche Richtung zukünftiger Recherchen. Auf diese Weise sollen unerwünschte Treffer bei zukünftigen Recherchen durch Berechnung von Filterbegriffen aus der Schale der geringwertigen Treffer vermieden werden. Durch die automatische Persistenz der Treffer unterstützt der Assistent den Schüler ebenfalls bei der Archivierung der erzielten Treffer und bietet durch die verschiedenen Darstellungen der Schaleninhalte gleichzeitig eine Strukturierungsunterstützung bei der Auswertung der Treffer.

Durch die Integration der LAssi-Volltextsuche ist es dem Assistenten möglich in anderen, möglicherweise bereits gelösten, Aufgabenstellungen zu recherchieren. Er kann den Schüler an dessen eigenes Wissen erinnern und ihn dadurch dabei unterstützen, seine bereits erarbeiteten Lösungen in neue Aufgabenstellungen einzubringen. Eine Desktop-Volltextsuche, wie z.B. Google Desktop, hat es schwer diesem Anspruch zu genügen, denn Google Desktop findet (nur) einzelne Dokumente auf dem lokalen Computer. Die LAssi-Suche ist zwar auch eine lokale Volltextsuche, genau wie Google Desktop, aber aufgrund des LAssi-Desktop-Konzeptes findet die LAssi-Suche Mengen von Dokumenten (LAssi-Desktops mit LAssi-Materialien), deren Inhalte in einer semantischen Beziehung zueinander stehen[2]. Hierfür sind keine semantischen Annotationen durch den Schüler notwendig. Er muss z.B. an keiner Stelle im LAssi-System angeben, dass ein bestimmter Karteikartenstapel zur “Garnelen”-Aufgabe gehört, sondern er legt den Stapel einfach auf dem zugehörigen Desktop an. Daraus ergibt sich automatisch eine semantische Verbindung. Weil der Assistent als Meta-Suchmaschine in der Lage ist, die Ergebnisse anderer Suchmaschinen in einer gemeinsamen Darstellung anzuzeigen, ist er in der Lage, dem Schüler Verknüpfungen seines Wissens mit extern verfügbarem Wissen aufzuzeigen.

[2]Die LAssi-Volltextsuche indiziert die XML-Repräsentationen der LAssi-Materialien auf dem USB-Stick und findet daher einzelne LAssi-Materialien. An den Java-Objekten dieser Materialien ist der zugehörige Desktop abfragbar.

5 Grundlagen der Clusteranalyse

Voraussetzung für die Clusteransicht einer Trefferschale des Assistenten ist die Berechnung dieser Cluster. Der Assistent muss die Clusterberechnung selbst vornehmen, da von den meisten Suchmaschinen keine Treffercluster abgerufen werden können. Dieses Kapitel enthält die für die Implementierung notwendigen mathematischen Grundlagen zur Berechnung von Clustern. Die Einbettung dieser Verfahren in den Assistenten wird in den folgenden Kapiteln konkretisiert.

5.1 Einführung in die Clusteranalyse

Gruppen von inhaltlich ähnlichen Objekten, die Teilmengen einer größeren Gesamtmenge sind, werden als Cluster bezeichnet[1]. Das Problem liegt im "Entdecken" der Cluster durch einen Algorithmus. Das Teilgebiet Clusteranalyse der Statistik beschäftigt sich mit der Entwicklung von Verfahren zur Lösung dieses Problems.

Die statistischen Verfahren der Clusteranalyse benötigen ein numerisches Datenfundament, auf dem sie operieren können. Formal betrachtet geht es um die Klassifikation von N Objekten $I = \{I_1, ..., I_N\}$. Jedes Objekt wird durch einen Merkmalsvektor $\vec{x} = (x_1, ..., x_p)$ mit p Merkmalen repräsentiert. Im konkreten Fall des Clustering auf Textbasis würde jede Dimension des Vektors $\vec{x}$, also jedes Merkmal, einen Begriff repräsentieren. Der Wert der zugehörigen Komponente des Vektors $\vec{x}$ enthält dann eine Häufigkeitsbewertung für den entsprechenden Begriff im zugehörigen Textobjekt. Die einzelnen Bewertungen sind als reelle Zahlen verhältnisskaliert, da sie sowohl über eine Ordnung, als auch über definierte Abstände und einen festen Nullpunkt (Begriff im Treffer nicht vorhanden) verfügen. Es ergibt sich folglich insgesamt eine Matrix X, deren Spalten die einzelnen Begriffe, deren Zeilen die zu klassifizierenden Treffer und deren Zellen die Bewertungen des jeweiligen Begriffs in einem Treffer repräsentieren[2]:

[1] vgl. [6], Seite 437
[2] vgl. [6], Seite 438 ff.

$$X = \begin{bmatrix} x_1 1 & \cdots & x_1 p \\ \vdots & & \vdots \\ x_N 1 & \cdots & x_N p \end{bmatrix}$$

Ein Beispiel für eine konkrete Ausprägung dieser Merkmalsmatrix vor dem Hintergrund des "Garnelen"-Beispiels zeigt die Abbildung Nr. 5.1.

	garnele	garnelenzucht	wattenmeer	nordsee	aquaristik
Treffer 1	1	2	12	1	1
Treffer 2	3	0	0	0	5
Treffer 3	7	1	9	19	6

Abbildung 5.1: Auszug aus einer Merkmalsmatrix für das "Garnelen"-Beispiel

5.2 Ähnlichkeits- und Distanzmaße

Alle Verfahren der Clusteranalyse nutzen zur Identifikation der Cluster ein Ähnlichkeits- oder Distanzmaß. Im Prinzip beschreiben beide Maße denselben Zusammenhang: Je größer die Ähnlichkeit zweier Objekte ist, desto geringer ist ihre Distanz zueinander. Umgekehr gilt, dass je größer die Distanz dieser zwei Objekte zueinander ist, desto unähnlicher sind sie sich. Kaufmann und Pape geben folgende formale Definition[3]:

Sei $I = I_1, ..., I_N$ eine Menge von N Objekten. Eine Funktion $s : I \times I \rightarrow \mathbb{R}$ heißt Ähnlichkeitsmaß, wenn gilt

$$s_{nm} = s_{mn} \tag{5.1}$$

$$s_{nm} \leqq s_{nn} \tag{5.2}$$

$$n, m = 1, ..., N$$

Die Definition verlangt von einer Ähnlichkeitsfunktion die Eigenschaft der Symmetrie in der Bedingung (5.1) (Objekt n ist Objekt m ebenso ähnlich, wie Objekt m Objekt n ähnlich ist) und dass die Ähnlichkeit eines Objektes m zu einem Objekt n nicht größer sein darf, als die Ähnlichkeit von n zu sich selbst (siehe 5.2). Kaufmann und Pape ergänzen, dass der Wertebereich vieler Ähnlichkeitsfunktionen auf das Intervall [0,1] beschränkt ist.

Ein in der Praxis sehr häufig verwendetes Ähnlichkeitsmaß ist das Cosinusmaß[4]. Dieses

[3] vgl. [6], Seite 440 ff.
[4] vgl. [5], Seite 191 ff. und [12], Seite 91 ff.

Maß betrachtet die Merkmalsvektoren zweier Objekte und bestimmt den Wert der Cosinus-Funktion des Winkel α zwischen beiden Merkmalsvektoren über die Formel (5.3)[5].

$$\cos\alpha = \frac{\vec{a}\cdot\vec{b}}{\left|\vec{a}\cdot\vec{b}\right|} = \frac{\sum_{i=1}^{p} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{p} a_i} \cdot \sqrt{\sum_{i=1}^{p} b_i}} \tag{5.3}$$

Die Idee des Cosinusmaßes ist, dass die Merkmalsvektoren ähnlicher Objekte einen sehr kleinen Winkel einschließen. Die Vorteile des Cosinusmaßes sind seine Anschaulichkeit und seine einfache Berechenbarkeit. Außerdem ist das Cosinusmaß für verhältnisskalierte Merkmale nutzbar. Der Nachteil des Cosinusmaßes ist, dass es die Streuung der Merkmalspunkte nicht berücksichtigt. Dies soll an einem kleinen Beispiel veranschaulicht werden:

Gegeben seien die Merkmalsvektoren $\vec{a} = (1,1,1)$ und $\vec{b} = (100,100,100)$. Der Winkel zwischen beiden den Vektoren beträgt 0°, es liegt also der maximale Ähnlichkeitswert vor. Das Problem liegt darin, dass der Vektor $\vec{b}$ ein Ausreißer sein, und dies gegen einen hohen Ähnlichkeitswert sprechen könnte. Es stellt sich die Frage, ob zwei Treffer wahrscheinlich in einen gemeinsamen Cluster gehören, wenn sie dieselben Begriffe enthalten, oder erst dann, wenn sie auch ungefähr gleich viele Vorkommen dieser Begriffe beinhalten. Da der Assistent eine inhaltlich relativ homogene Menge von Treffern recherchieren wird, werden alle Treffer ein ähnliches Vokabular verwenden. Daher ist für die Berechnung von Clustern ein Maß erforderlich, welches auch die Häufigkeit eines Begriffs in einem Treffer berücksichtigt. Wenn man die Merkmalsvektoren als Punkte im Raum interpretiert, dann interessiert für das Clustering deren tatsächlicher Abstand zueinander.

Nach den Ähnlichkeitsmaßen sollen auch Distanzmaße formal definiert werden. Eine Funktion $d : I \times I \rightarrow \mathbb{R}$ heißt Distanzmaß, wenn gilt[6]

$$d_{nn} = 0 \text{ und } d_{nm} \geqq 0 \tag{5.4}$$

$$d_{nm} = d_{mn} \tag{5.5}$$

$$n,m = 1,...,N$$

Ein Distanzmaß verfügt ebenfalls über die Eigenschaft der Symmetrie (siehe 5.5). Außerdem hat ein Objekt n zu sich selbst die Distanz 0 und eine Distanz darf nie negativ sein (siehe 5.4). Der angesprochene Zusammenhang zwischen Ähnlichkeits- und Distanzfunktionen wird durch (5.2) und (5.4) deutlich. Ein in der Literatur sehr häufig genanntes Distanzmaß ist die

[5] siehe [7], Seite 60 ff.
[6] vgl. [6], Seite 440 ff.

euklidische Distanz[7]. Die euklidische Distanz d zweier Merkmalspunkte x_n und x_m berechnet sich wie folgt:

$$d(x_n, x_m) = \sqrt{\sum_{i=1}^{p} (x_{ni} - x_{mi})^2} \tag{5.6}$$

Das Verfahrens berechnet den Abstand zweier Punkte im mehrdimensionalen Raum. Dazu werden dimensionsweise die Distanzen der jeweiligen Stellen ermittelt, um abschließend einen mittleren Abstand aus der Summe aller berechneten Dimensionsabstände zu erhalten. Die euklidische Distanz ist wie auch das Cosinusmaß auf verhältnisskalierte Merkmale anwendbar. Ihr Vorteil gegenüber dem Cosinusmaß ist, dass die euklidische Distanz die Streuung der Treffer berücksichtigt. Dieser Sachverhalt ist beispielhaft in der Abbildung 5.2 für den zweidimensionalen Raum dargestellt.

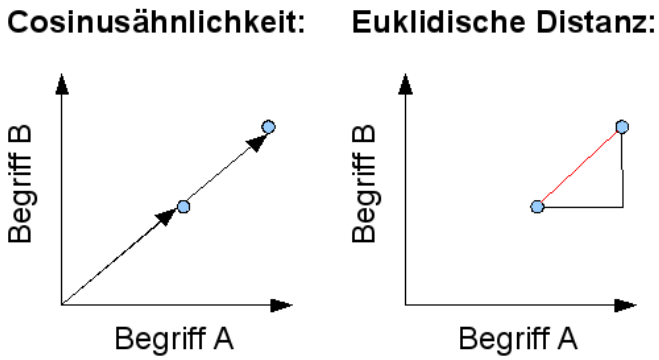

Abbildung 5.2: Cosinusmaß und Euklidische Distanz im zweidimensionalen Raum

Die zwei Dimensionen stehen jeweils für einen Begriff. Die eingetragenen Punkte bzw. Vektoren repräsentieren zwei Treffer, welche beide Begriffe unterschiedlich oft enthalten. Beide dargestellten Vektoren erhalten durch das Cosinusmaß den maximalen Ähnlichkeitswert, denn es ist erkennbar, dass die zwei Vektoren direkt aufeinander liegen und damit einen Winkel von 0°umschließen (siehe obiges Negativbeispiel zum Cosinusmaß). Die euklidische Distanz hingegen liefert einen Abstandswert, der anschaulich dem natürlichen Empfinden eines Abstands entspricht. Die euklidische Distanz entspricht im Zweidimensionalen dem Satz des Pythagoras und liefert damit die Länge der Strecke zwischen den zwei Punkten (rot dargestellt). Damit erscheint sie für die Berechnung der Treffercluster besser geeignet als das Cosinusmaß.

[7] vgl. [6], Seite 448 ff. und [9], Seite 36 ff.

5.3 Anwendung von Ähnlichkeits- und Distanzmaßen auf Cluster

Bei der Anwendung des im nachfolgenden Kapitel vorgestellten Verfahren tritt das Problem der Bestimmung von Distanzen bzw. Ähnlichkeiten zwischen zwei Clustern auf. Die vorgestellten Distanz- bzw. Ähnlichkeitsmaße können jedoch nur auf zwei Vektoren bzw. Punkte angewendet werden, Cluster sind aber Mengen von Vektoren bzw. Punkten. Es existieren in der Literatur mehrere Strategien zur Lösung dieses Problems. An dieser Stelle soll eine sehr häufig erwähnte und implementierte Idee vorgestellt werden[8]. Es besteht die Möglichkeit, einen Vektor bzw. Punkt zu bestimmen, der den Cluster repräsentiert ("Repräsentant"). Auf derlei Vektoren bzw. Punkte können die vorgestellten Distanz- und Ähnlichkeitsmaße angewendet werden. Ein solcher Ansatz ist das Zentroid-Verfahren, welches den Zentralvektor bzw. Mittelpunkt eines Clusters (beides wird Zentroid genannt) berechnet und anhand dieses Zentroiden den Abstand des Clusters zu anderen Vektoren oder Clustern berechnet. In der Abbildung 5.3 sind zwei Cluster im zweidimensionalen Raum mitsamt ihrer Zentroiden dargestellt. Der Abstand der Cluster entspricht dem Abstand der Zentroiden zueinander (rot markierte Strecke).

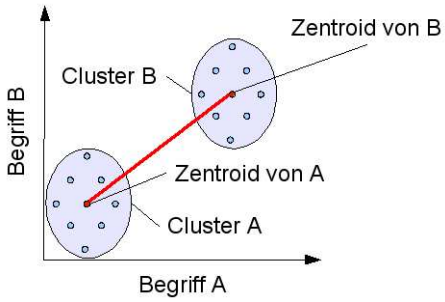

Abbildung 5.3: Cluster mit zugehörigem Zentroiden im zweidimensionalen Raum

5.4 Vorstellung des k-Means-Verfahrens

Ein häufig implementiertes Clustering-Verfahren ist das k-Means-Verfahren. Das k-Means-Verfahren ist ein partitionierendes Clustering-Verfahren. Partitionierende Verfahren teilen eine gegebene Menge von Objekten in k Gruppen (Partitionen) auf, sodass das Ergebnis dieser Verfahren eine optimale Verteilung aller Objekte auf die k Partitionen darstellt. Der Wert k wird diesem Verfahren als Parameter mitgeteilt. Innerhalb der k Partitionen sind nach Abschluss eines partitionierenden Verfahrens keine weiteren Partitionen berechnet[9].

[8] vgl. [12], Seite 107 ff. und [6], Seite 460 ff.
[9] vgl. [12], Seite 109 ff.

Dieses Verfahren erhält neben dem Parameter k die zu clusterende Objektmenge und ordnet die einzelnen Objekte den k Clustern solange zu, bis die optimale Zuordnung gefunden worden ist. Die Abbildung Nr. 5.4 stellt das Prinzip des k-Means-Verfahrens dar[10].

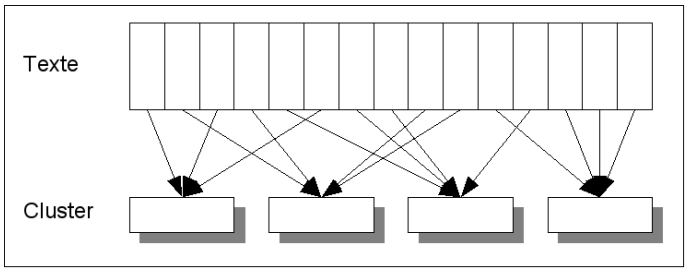

Abbildung 5.4: Das Prinzip des k-Means-Verfahren

Weiss et al. geben den k-Means-Algorithmus wie folgt an[11]:

1. Distribute all documents among the k bins.
2. Compute the mean vector for each bin.
3. Compare the vector of each document to the bin means and note the mean vector that is most similar.
4. Move all documents to their most similar bins.
5. If no document has been moved to a new bin, then stop; else go to step 2.

Das Verfahren durchläuft zwei Phasen: eine Initialisierungsphase und eine Clusteringphase. In der Initialisierungsphase werden k leere Cluster erstellt und die zu clusterenden Objekte auf diese leeren Cluster verteilt. Prinzipiell kann die Verteilung willkürlich erfolgen. Weiss et al. schlagen jedoch eine Optimierung vor, um die Anzahl der Iterationen des Verfahrens zu verringern. Die Optimierung besagt, einen Zentroiden über alle zu clusterenden Objekte zu berechnen. Im Anschluss daran ist mit einem Ähnlichkeits- bzw. Distanzmaß die Ähnlichkeit oder Distanz eines jeden Vektors zum Zentroiden zu bestimmen und in einer Liste zu erfassen. Abschließend wird diese Liste absteigend nach Ähnlichkeiten bzw. aufsteigend nach Distanzen sortiert und in k Abschnitte geteilt. Diese k Abschnitte entsprechen den Initialisierungsclustern.

Es beginnt nun die Clusteringphase im ersten Schritt des Verfahrens mit der Berechnung der Repräsentanten (“mean vector”) für jeden der k Cluster. Im Anschluss daran wird für

[10] Entnommen und geringfügig modifiziert aus [12], Seite 110
[11] vgl. [12], Seite 109 ff.

jeden Vektor der k Cluster die Ähnlichkeit bzw. die Distanz zu allen k Repräsentanten berechnet. Danach verschiebt das Verfahren jeden Vektor in den Cluster des ähnlichsten bzw. des dichtesten Repräsentanten, es sei denn, der Vektor befindet sich bereits dort. Wenn keine Vektoren mehr verschoben worden sind, ist die optimale Lösung erreicht. Ansonsten beginnt das Verfahren erneut mit der Berechnung der neuen Repräsentanten, um eine neue Iteration zu beginnen.

Das k-Means-Verfahren wird in der Literatur für seine Anschaulichkeit und seine guten Ergebnisse gelobt[12]. Ein weiterer großer Vorteil des Verfahrens ist seine lineare Laufzeitkomplexität. Als Nachteil des Verfahrens ist zu nennen, dass die Anzahl der zu berechnenden Cluster vorgegeben werden muss. Zu Beachten ist in der Implementierung des Verfahrens, dass das Verfahren auch leere Cluster zurückliefern kann.

Für den konkreten Einsatz im Rahmen des Assistentensystems eignen sich nur Verfahren, die disjunkte Cluster erzeugen, also jeden Treffer in genau einen Cluster einteilen[13]. Eine unscharfe Klassifikation, also die Einordnung eines Treffers in mehrere Cluster, ist nicht betrachtet worden, da dies höchstwahrscheinlich zur Unübersichtlichkeit der Treffermenge führen wird. Hierarchische Clustering-Verfahren wurden ebenfalls nicht betrachtet, weil diese mit quadratischer Laufzeitkomplexität arbeiten.

Mit dem k-Means-Verfahren können Cluster von Treffern auf Basis deren Textinhalte berechnet werden. Die Abbildung 5.5 zeigt diesen Prozess anhand des “Garnelen”-Beispiels. Die aus den Treffern der Trefferschale berechneten Cluster werden durch die Clusteransicht des Assistenten dargestellt. Bevor die Cluster dort angezeigt werden, wird zusätzlich noch eine Bezeichnung (Label) für die Cluster berechnet.

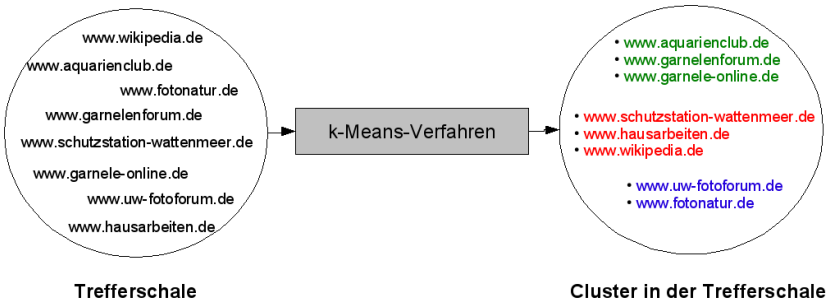

Abbildung 5.5: Anwendung des k-Means-Verfahren zur Berechnung von Trefferclustern

[12] vgl. [12], Seite 109 ff.
[13] vgl. [12], Seite 73

6 Statistische Bewertungsverfahren für Deskriptoren

Dieses Kapitel stellt Verfahren zur Bestimmung von charakteristischen Begriffen aus einer Menge von Dokumenten bzw. Treffern vor. Die Verfahren basieren auf statistischen Häufigkeiten. Diese Begriffe werden als Deskriptoren bezeichnet[1]. Ein Deskriptor ist entweder ein einzelner Begriff (z.B. “Wattenmeer”) oder eine Kombination einer bestimmten Anzahl von Begriffen (z.B. “Lebensraum Wattenmeer”). Die maximale Anzahl ist vom Schüler festzulegen. Die vorgestellten Verfahren sind für den Assistenten an vier Stellen notwendig:

1. Auswahl von Suchbegriffen aus dem aktuell geöffneten LAssi-Desktop und den Schaleninhalten der hochwertigen, zu berücksichtigenden Treffer
2. Auswahl von Filterbegriffen aus den Schaleninhalten der geringwertigen Treffer
3. Auswahl der Spaltenbezeichnungen für die Merkmalsmatrix X zur Clusterberechnung der Treffer aus der vom Schüler selektierten Schale
4. Berechnung von Labels für die aus den Treffern einer Schale berechneten Cluster

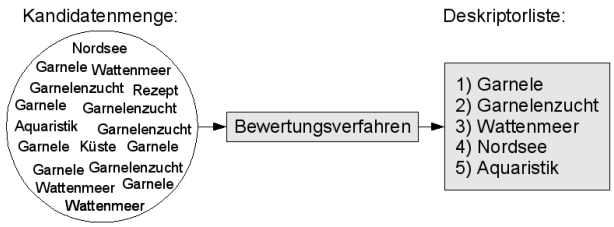

Abbildung 6.1: Auswahl von fünf Deskriptoren

Voraussetzung für die vorgestellten Verfahren ist eine Menge von Kandidaten (siehe Abbildung 6.1). Diese Menge ist eine Multimenge, d.b. dass sie ein Element mehrfach enthalten kann. Die Berechnung dieser Kandidatenmenge und deren software-technische Umsetzung

[1] vgl. [3] , Kapitel 1.3.4 - Thesauren

wird im Kapitel zum Systementwurf beschrieben. Ebenso wird dort die Einbettung der an dieser Stelle vorgestellten Verfahren in den Assistenten konkretisiert. Die Verfahren werden anhand der zwei LAssi-Karteikarten aus der Abbildung 6.2 veranschaulicht. Dabei gelte, dass die Titel mit einer Gewichtung von 6,0 Punkten und die Rümpfe mit 4,0 Punkten bewertet werden.

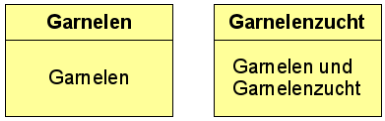

Abbildung 6.2: Beispiele für LAssi-Karteikarten

6.1 Absoluter Häufigkeitsansatz

Der absolute Häufigkeitsansatz wertet die absolute Häufigkeit eines Kandidaten aus. Es gilt hier die Annahme, dass sich die Relevanz eines Kandidaten proportional zu seiner Häufigkeit verhält. Weiss et al. geben ein mögliches Verfahren zur Bestimmung der Häufigkeiten an[2]. Dieser Ansatz hat den Vorteil, dass er anschaulich und seine Statistik mit linearer Laufzeitkomplexität zu berechnen ist. Es ist zu beachten, dass die häufigsten Kandidaten einer Menge von Dokumenten die gemeinsamen Kandidaten der Dokumente repräsentieren, also nicht charakteristisch für einzelne Dokumente sind. Dies kann sich in bestimmten Problemstellungen als nachteilhaft erweisen, z.B. bei der Berechnung von Spaltenbezeichnungen einer Merkmalsmatrix für ein Clustering-Verfahren. Diese Verfahren benötigen für brauchbare Ergebnisse Kandidaten, die die Dokumente gut voneinander abgrenzen. Zur Berücksichtigung der Gewichtung der Position eines Kandidaten im Dokument, wird diese multiplikativ mit der absoluten Häufigkeit verknüpft, sodass sich für die Bewertung des Kandidaten j über n Dokumente ergibt:

$$h(j) = \sum_{i=1}^{n} \sum_{p=1}^{dp} tf(j,i) \cdot weight(j,i,p) \tag{6.1}$$

Die Funktion $tf(j,i)$ liefert die absolute Häufigkeit des Kandidaten j im Dokument i und die Funktion $weight(j,i,p)$ das Gewicht der Position p des Kandidaten j im Dokument i. dp sei die Anzahl der gewichteten Positionen des jeweiligen Dokumentes. Die Begriffe der Beispielkarteikarten würden wie folgt bewertet: $h(Garnelen) = 6.0 \cdot 1 + 4.0 \cdot 2 = 14.0$ Punkte und $h(Garnelenzucht) = 6.0 \cdot 1 + 4.0 \cdot 1 = 10$ Punkte. Daher wird der Begriff “Garnele” dem Begriff “Garnelenzucht” vorgezogen.

[2] siehe [12], Seite 26

6.2 tf-idf-Häufigkeitsansatz

Das tf-idf-Verfahren ist von Gerard Salton entwickelt worden und findet sowohl in der Literatur, als auch in der Praxis großen Anklang[3]. Die Idee des Verfahrens ist, dass ein guter Kandidat für einen Deskriptor des Dokuments A innerhalb von A sehr häufig vorkommt und in anderen Dokumenten sehr selten enthalten ist. Dann scheint dieser Begriff für A charakteristisch zu sein. Das Verfahren basiert auf absoluten Häufigkeiten, welche durch einen Korrekturfaktor korrigiert werden sollen, um den angesprochenen Nachteil der absoluten Häufigkeiten zu kompensieren. Das tf-idf-Maß für einen Kandidaten j berechnet sich wie folgt:

$$tfidf(j) = tf(j) \cdot idf(j) = tf(j) \cdot \frac{1}{df(j)} \tag{6.2}$$

Die Funktion $tf(j)$ liefert die absolute Häufigkeit des Kandidaten j über alle Dokumente und die Funktion $df(j)$ die Anzahl der Dokumente, in denen der Kandidat j vorkommt. Die Funktion idf(j) (inverse document frequency) invertiert die gelieferte Anzahl der Dokumente. Auf diese Weise wird die Bewertung von Kandidaten, die in vielen Dokumenten vorkommen niedriger angesetzt, als die von Kandidaten, die in wenigen Dokumenten enthalten sind. Weiss et al. schlagen eine andere Variante der Funktion $idf(j)$ vor:

$$idf(j) = log(\frac{N}{df(j)}) \tag{6.3}$$

N ist dabei die Anzahl der Dokumente, die insgesamt betrachtet werden. Mit der Logarithmierung wird eine Einschränkung des Ergebnisbereiches erreicht. Zur Berücksichtigung der Gewichtungen einzelner Positionen wird statt der einfachen absoluten Häufigkeit $tf(j)$ die gewichtete Häufigkeit $h(j)$ verwendet, wie folgende formale Notation zeigt:

$$tfidf(j) = h(j) \cdot idf(j) \tag{6.4}$$

Die Begriffe “Garnelen” und “Garnelenzucht” würden wie folgt bewertet: $tfidf(Garnelen) = 14.0 \cdot \frac{1}{2} = 7.0$ Punkte und $tfidf(Garnelenzucht) = 10.0 \cdot \frac{1}{1} = 10.0$ Punkte. Dieses Verfahren bevorzugt den Begriff “Garnelenzucht”.

[3] vgl. [10], Seite 68 und [5], Seite 78 ff.

7 Systementwurf

7.1 Übersicht

Der interne Datenfluss und die Verarbeitungsschritte des zu entwickelnden Assistenten sind in der Abbildung 7.1 dargestellt.

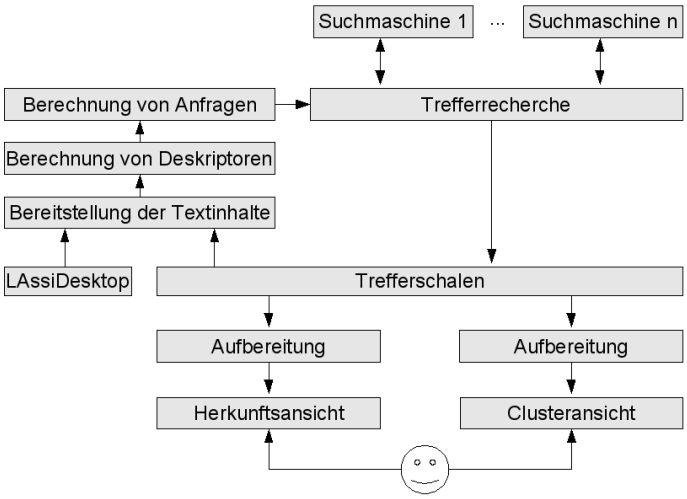

Abbildung 7.1: Systementwurf für den Assistenten

Ein Rechercheprozess des Assistenten beginnt mit der Bereitstellung der Textinhalte aller LAssi-Materialien und deren Dateianhänge auf dem aktuellen LAssi-Desktop. Außerdem werden die Textinhalte der hochwertigen, zu berücksichtigenden und der geringwertigen Treffer aus den entsprechenden Schalen benötigt. Dazu müssen die enthaltenen String-Daten von spezifischen Formatangaben getrennt werden, z.B. durch Filterung aller HTML-Tags mitsamt ihren Attributen aus einer HTML-Seite, welche durch Yahoo gefunden worden ist.

Anschließend werden aus den bereitgestellten Strings Deskriptoren (Such- und Filterbegriffe) berechnet. Dazu werden die Strings anhand der Leer- und Satzzeichen in Listen von Wörtern getrennt. Diese Wörter werden als Token bezeichnet. Danach versucht der Assistent diese Listen u.a. durch Filtern von nichtssagenden Wörtern (z.B. bestimmte oder unbestimmte Artikel im Deutschen) zu reduzieren. Diese nichtssagenden Wörter werden als Stoppwörter

bezeichnet. Abschließend bewertet der Assistent mit den vorgestellten statistischen Häufigkeitsansätzen die Token und selektiert aus den bewerteten Token Deskriptoren.

Aus den berechneten Deskriptoren generiert der Assistent im Folgenden die Anfragen. Dazu werden alle Kombinationen aus Such- und Filterbegriffen berechnet und absteigend nach Strenge geordnet. Die erste berechnete Suchanfrage wird jetzt an die integrierten Suchmaschinen gesendet und deren Treffer empfangen. Ist die vom Schüler gewünschte Trefferanzahl durch diese Anfrage nicht erzielt worden, so wird die nächst schwächere Anfrage gesendet. Aus den resultierenden Treffern wird die noch benötigte Anzahl selektiert. Dies geschieht solange, bis die definierte Trefferanzahl recherchiert worden ist oder keine weiteren Anfragen mehr vorhanden sind. Jedes Trefferdokument soll dabei nur einziges Mal berücksichtigt werden. Die erzielten Treffer werden jetzt in der Sammelschale gespeichert. Nach Abschluss aller Recherchen erhält der Schüler eine Hinweismeldung am Bildschirm.

Er kann die erzielten Treffer nun in jeder beliebigen Schale mit der Herkunfts- oder der Clusteransicht ansehen. Dazu wird die selektierte Trefferschale aufbereitet, z.B. durch Berechnung von Clustern in Abhängigkeit zum Trefferinhalt (Clusteransicht) oder zur Quellsuchmaschine (Herkunftsansicht). Der Schüler hat an dieser Stelle die Möglichkeit die Treffer zu öffnen, auszuwerten und durch Verschieben in andere Schalen zu bewerten. Nach einer vom Schüler definierten Zeitspanne beginnt der Assistent den Rechercheprozess auf Basis des aktuellen LAssi-Desktop und der Schaleninhalte von vorne.

7.2 Bereitstellung von LAssi-Desktop- und Schaleninhalten

Der Assistent benötigt für die Berechnung der Kandidatenmenge für Such- und Filterbegriffe Zeichenketten aus dem Arbeitskontext des Schülers. Wie bereits erläutert, sollen diese aus den Inhalten des aktuellen LAssi-Desktops, aus der Schale der hochwertigen, zu berücksichtigenden und der Schale der geringwertigen Treffer gewonnen werden. Das Problem ist, dass die LAssi-Materialien, deren Anhänge und auch die Treffer völlig unterschiedliche Datenformate besitzen können. Beispielweise speichert ein LAssi-Material seinen Zustand in einem XML-Format, sein Anhang enthalte ein Word- und ein PDF-Dokument und die geringwertigen Treffer seien allesamt von der Suchmaschine Yahoo gefundene HTML-Seiten. Aufgrund der Vielzahl der möglichen Datenformate kann der Assistent die Zeichenketten unmöglich selbst auslesen, sondern ist auf Konvertoren angewiesen. Außerdem ist es dem Assistenten nicht möglich, der Position einer Zeichenkette im Dokument eine Gewichtung zuzuordnen, die deren semantische Rolle repräsentiert. Beispielsweise sollen Token des Titels einer LAssi-Karteikarte stärker gewichtet werden, als Token aus den Texteinträgen im Rumpf der Karteikarte. Die Gewichtung von Positionen in einem Dokument soll, wie bereits angedeutet, in

der Berechnung der Häufigkeitsstatistik der Token berücksichtigt werden. Daher müssen diese Konvertoren von dem Entwickler bereitgestellt werden, der ein neues Format in das System einführt, z.B. durch ein neues LAssi-Material.

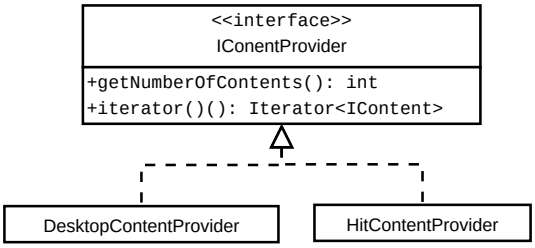

Abbildung 7.2: UML Klassendiagramm für die Schnittstelle “IContentProvider”

Ein Konvertor entspricht im Assistenten einem “IContentProvider”. Die Abbildung 7.2 enthält das zugehörige UML Klassendiagramm. Eine Instanz von “IContentProvider” ist zuständig für die Bereitstellung der Textinhalte einer Menge von Dokumenten, z.B. der Inhalte eines LAssi-Desktops oder einer Trefferschale. Die Methode “getNumberOfContents()” liefert die Anzahl der Elemente dieser Menge. Diese Anzahl wird für die Berechnung der Arbeitsschritte des gesamten Ausleseprozesses benötigt, mit welcher u.a. der Fortschrittsbalken der zum Prozess gehörenden Statusanzeige initialisiert wird.

Über die Methode “iterator()” ist es möglich, ein Iteratorobjekt für die Dokumentenmenge abzurufen. Mithilfe eines Iteratorobjektes kann eine Datenstruktur traversiert werden, ohne deren Implementierung zu kennen, z.B. die Menge der vom LAssi-Desktop aus erreichbaren LAssi-Materialien (transitive Desktophülle) oder die Treffer aus einer Schale. Das Klassendiagramm eines Iterators ist in der Abbildung 7.3 dargestellt[1].

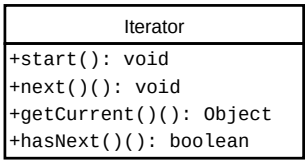

Abbildung 7.3: UML Klassendiagramm für einen Iterator

Die Methode “start()” setzt den Iterator auf das erste Element der Datenstruktur. Über die Methode “getCurrent()” ist das aktuelle Element der Datenstruktur zugreifbar. “hasNext()” prüft, ob ein weiteres Element in der Struktur existiert und “next()” weist den Iterator an, intern das nächste Element der Struktur zu referenzieren. Der vom “IContentProvider” gelieferte Iterator referenziert entweder ein LAssi-Material oder einen Treffer. Es ist mit ihm folglich

[1] vgl. [4], Seite 335 ff.

möglich, die Dokumentmenge des “IContentProvider” zu traversieren. Wie kann der Assistent nun den Textinhalt des LAssi-Materials bzw. des Treffers abzurufen?

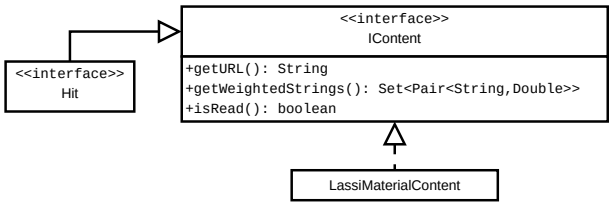

Abbildung 7.4: UML Klassendiagramm der Schnittstelle “IContent”

Für diesen Zweck ist die Schnittstelle “IContent” definiert worden (siehe Abbildung 7.4). Die diese Schnittstelle implementierenden Klassen nehmen die Rolle eines Adapters ein. Sie bilden intern die Funktionalität der adaptierten Schnittstelle auf die eigene ab, z.B. die eines LAssi-Materials.[2]. Alle Objekte, die der Iterator eines “IContentProvider” liefert, sind Instanzen einer diese Schnittstelle implementierenden Klasse. Ein solches Objekt verfügt über die Methode “getURL()”, mit der der URL (Uniform Resource Locator) des jeweiligen Dokuments abgerufen werden kann. Ein URL ist innerhalb eines bestimmten Namensraumes (z.B. das World Wide Web) ein eindeutiger Bezeichner einer Resource, z.B. einer Datei auf einem Webserver oder auf einer lokalen Festplatte. Der Assistent verwendet den URL der LAssi-Materialien und seiner Treffer, um doppelte Treffer zu erkennen und auszuschließen. Die eigentlichen Textinhalte lassen sich über die Methode “getWeightedStrings()” abrufen. Diese Methode liefert eine Menge von Paaren aus einer Zeichenkette und einer Fließkommazahl. Ein Paar entspricht einem mathematischen Tupel. Es wird repräsentiert durch eine Klasse mit zwei Membervariablen vom allgemeinsten Datentyp (z.B. java.lang.Object in Java) und jeweils einer Getter- und Setter-Methode für jede Variable. Eine Membervariable ist eine globale Variable einer Klasse, deren Werte an die verschiedenen Instanzen der Klasse gebunden sind. Die Fließkommazahl repräsentiert das Gewicht der Zeichenkette im gesamten Dokument. Daher ist das Resultat der Operation eine Menge von gewichteten Zeichenketten.

Die Gewichte müssen vom Entwickler derjenigen Klasse festgelegt werden, die die Schnittstelle “IContent” implementiert. Um zu verhindern, dass Entwickler “ihre” Dokumente besonders stark gewichten können, ist ein Punktwertmodell definiert worden. Die Summe aller Gewichtungen muss gleich 10.0 Punkten sein - wie der Entwickler diese Punkte unter einzelnen Positionen “seines” Datenformates verteilt, bleibt ihm überlassen. Bevor ein “IContent”-Objekt vom Assistenten ausgewertet werden darf, wird diese Invariante (Gewichtungssumme = 10.0) geprüft. Bei Verletzung der Invariante wird das betreffende “IContent”-Objekt übersprungen.

[2] vgl. [4], Seite 171

7.3 Aufbereitung und statistische Analyse der Textinhalte

Aus den bereitgestellten gewichteten Zeichenketten soll eine Menge von Kandidaten für Deskriptoren berechnet werden. Der Assistent benötigt diese Funktionalität zur Berechnung von Such- und Filterbegriffen und zur Benennung der Cluster einer Schale für die Clusteransicht. Außerdem werden auf diese Weise die Spaltenbezeichnungen der Merkmalsmatrix X bestimmt, welche zur Berechnung von Clustern aus den Inhalten von Trefferschalen notwendig ist.

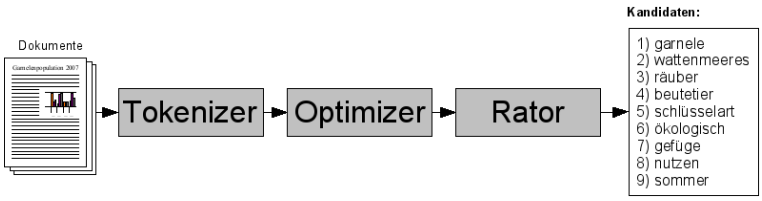

Abbildung 7.5: Verarbeitungskette zur Generierung von Kandidaten aus Zeichenketten

Die Abbildung 7.5 zeigt eine schematische Verarbeitungskette, welche aus den bereitgestellten Zeichenketten eine Liste von Deskriptorkandidaten generiert. Dazu werden die bereitgestellten Zeichenketten mitsamt ihrer Gewichtung dem Tokenizer als Parameter übergeben. Der Tokenizer hat nun die Aufgabe die Zeichenketten von Satz- und Steuerzeichen, sowie Klammern etc. zu bereinigen Dazu ersetzt er all diese Zeichen durch das Leerzeichen, da dieses Zeichen üblicherweise als Trennzeichen zwischen Wörtern verwendet wird. Anschließend wird die Zeichenkette in eine einheitliche Schreibweise (nur Gross- oder Kleinbuchstaben) konvertiert und anhand des Leerzeichens in gewichtete Listen von enthaltenen Wörtern (Token) gesplittet. Eine Liste gilt immer für einen Gewichtungsraum. Die Abbildung 7.6 veranschaulicht diesen Prozess anhand einer Karteikarte vom LAssi-Desktop. Der Titel einer Karteikarte wird mit 6.0 Punkten gewichtet und die Texteinträge mit 4.0 Punkten.

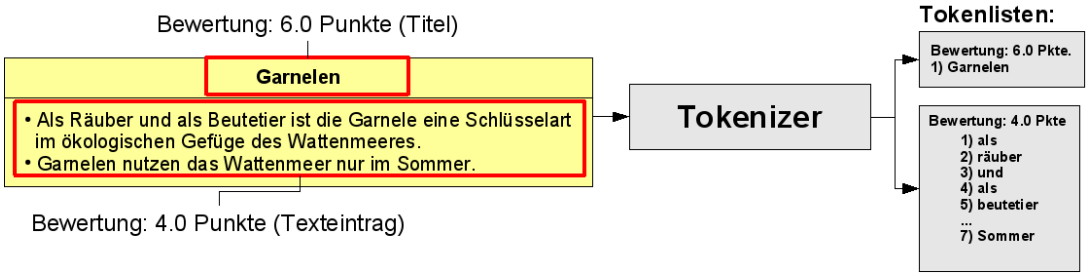

Abbildung 7.6: Aufspaltung von Zeichenketten durch den Tokenizer

Die Aufgabe des Optimizers besteht in der Reduktion der erhaltenen Tokenlisten. Dazu

werden die Stoppwörter pauschal aus den Listen entfernt, z.B. alle bestimmten und unbestimmten Artikel im Deutschen. Token mit einer Länge von weniger als drei Zeichen werden ebenfalls pauschal gefiltert, weil sie in der Regel als Suchbegriffe ungeeignet sind. Häufig treten dieselben Wörter in verschiedenen Schreibweisen auf, z.B. ein Verb in verschiedenen Zeitformen oder ein Nomen in verschiedenen Fällen. Algorithmen zur Erkennung einer gemeinsamen Grundform dieser Token werden als Stemming-Verfahren bezeichnet. Stemming-Verfahren lassen sich als statisch oder dynamisch klassifizieren. Ein statisches Verfahren ist z.B. das Nachschlagen einer Grundform eines Token in einem Thesaurus. Thesauren müssen allerdings erst einmal angelegt werden und sind aufwendig zu pflegen[3]. Eine Alternative ist das Porter-Stemming-Verfahren, welches zur Laufzeit gemeinsame Grundformen von Token anhand grammatikalischer Regeln der Sprache, aus der das Token stammt, berechnet [4]. Das Problem ist hier, dass bekannt sein muss, aus welcher Sprache ein Token stammt, um den korrekten Regelsatz für den Algorithmus auswählen zu können.

Gut einsetzbar wäre hingegen ein Verfahren, welches unabhängig von der Kenntnis der konkreten Sprache die Ähnlichkeit zweier Token messbar quantifiziert. Anhand dieser Ähnlichkeit kann dann entschieden werden, ob eines der zwei Token durch das andere repräsentiert werden kann. Diese Anforderung wird durch das n-Gram-Verfahren in Kombination mit dem Dice-Quotienten umgesetzt. Dieses Verfahren spaltet jedes der zwei zu vergleichenden Token in Subtoken mit der Länge n Zeichen, sogenannte n-Gramme. Alle Subtoken eines Token werden in einer Menge erfasst, sodass nach Abschluss der Subtoken-Bildung zwei Mengen von n-Grammen existieren. Diese zwei Mengen werden als Parameter T_a und T_b in die Dice-Funktion gegeben:

$$d(T_a, T_b) = \frac{2 \cdot |T_a \cap T_b|}{|T_a| + |T_b|} \tag{7.1}$$

Dieses Verfahren soll anhand eines kleinen Beispiels veranschaulicht werden. Betrachtet werden die Zeichenketten a = “garnele” und b = “nordseegarnele” für n=2 (Bigramme). Als Mengen ergeben sich $T_a = \{$ga, ar, rn, ne, el, le$\}$ und $T_b = \{$no, or, rd, ds, se, ee, ga, ar, rn, ne, el, le$\}$ mit der Schnittmenge $T_a \cap T_b = \{$ga, ar, rn, ne, el, le$\}$. Aus den Mengen T_a und T_b berechnet sich der Dice-Quotient wie folgt:

$$d(T_a, T_b) = \frac{2 \cdot 6}{6 + 12} = 0,\overline{6}$$

Das Dice-Maß beziffert die Ähnlichkeit der Zeichenketten “garnele” und “nordseegarnele” auf $\sim 66,67\%$. Eine gewisse Abhängigkeit zur Sprache der Token ist noch immer gegeben,

[3] vgl. [3] , Kapitel 1.3.4 - Thesauren
[4] vgl. [8], Kapitel 1 - Introduction

Token	Menge von Paaren aus (Gewicht,Häufigkeit)	Dokumente
räuber	(4.0,1)	KK “Garnelen”
beutetier	(4.0,1)	KK “Garnelen”
garnele	(6.0,1);(4.0,2)	KK “Garnelen”
schlüsselart	(4.0,1)	KK “Garnelen”
ökologischen	(4.0,1)	KK “Garnelen”
gefüge	(4.0,1)	KK “Garnelen”
wattenmeeres	(4.0,2);(6.0,1)	KK “Garnelen”; KK “Wattenmeer”
nutzen	(4.0,1)	KK “Garnelen”
sommer	(4.0,1)	KK “Garnelen”

Tabelle 7.1: Beispielhafte Tokenstatistik für zwei Karteikarten (KK)

denn über die Schreibweise kann beispielsweise nicht die Ähnlichkeit zum englischen Begriff “Shrimp” gemessen werden. Das Verfahren liefert hier eine Ähnlichkeit von 0%, dabei sind die Begriffe “Shrimp” und “Garnele” hochgradig ähnlich. Da aber sowohl der Begriff “Shrimp”, als auch “Garnele” als Suchbegriff zum Thema “Garnelen” gute Treffer erzielen wird, ist es gar nicht sinnvoll, einen der beiden Begriffe durch den anderen ersetzen zu lassen, sondern beide z.B. als Suchbegriffe zu verwenden. Durch die Nutzung dieses Verfahrens kann auf die Implementierung einer Sprachidentifkation für Texte verzichtet werden. Weil das Verfahren gute Ergebnisse erzielt, erscheint die n-Gram-Analyse das am besten für den Prototypen des Assistenten geeignete Verfahren zur Tokenlistenreduktion zu sein[5].

Neben der Tokenlistenreduktion soll der Optimizer die Tokenlisten zu einer Relation zusammenführen. Diese Relation wird dann den statistischen Bewertungsverfahren für Token übergeben. Einem Token soll hier eine Liste von Paaren aus seinen Häufigkeiten und den Gewichtungen seiner Positionen im Dokument zuordnet werden. Zusätzlich wird zu jedem Token noch die Anzahl der Dokumente gespeichert, in denen es enthalten ist. Ein Beispiel für diese Relation ist in der Tabelle 7.1 auf Basis des vorherigen Karteikartenbeispiels und einer zusätzlichen Karteikarte mit dem Titel “Wattenmeer” ohne weitere Texteinträge angegeben. Für diese Relation sind nur atomare Token herangezogen worden. Es soll für den Schüler aber möglich sein, eine maximale Anzahl von Token in einer Tokenkombination zu definieren. Die Statistik enthält dann auch Tokenkombinationen mit dieser maximalen Länge. Über die Kombination von Token werden strengere Anfragen erreicht, denn in den Treffern müssen die Token exakt aufeinanderfolgend in der Reihenfolge enthalten sein, die durch die Tokenkombination vorgegeben wird. Es wird angenommen, dass die gelieferten Treffer strengerer Suchanfragen präziser zur Aufgabenstellung des Schülers passen. Der Assistent könnte so als Suchbegriff beispielsweise die Tokenkombination “lebensraum wattenmeer” berechnen.

[5] vgl. [5], Seite 357 ff.

Rang	Kandidat	Bewertung
1.	garnele	$1 \cdot 6.0 + 2 \cdot 4.0 = 14.0$
2.	wattenmeeres	$2 \cdot 4.0 + 1 \cdot 6.0 = 14.0$
3.	räuber	$1 \cdot 4.0 = 4.0$
4.	beutetier	$1 \cdot 4.0 = 4.0$
5.	schlüsselart	$1 \cdot 4.0 = 4.0$
6.	ökologisch	$1 \cdot 4.0 = 4.0$
7.	gefüge	$1 \cdot 4.0 = 4.0$
8.	nutzen	$1 \cdot 4.0 = 4.0$
9.	sommer	$1 \cdot 4.0 = 4.0$

Tabelle 7.2: Beispielhafte Kandidatenliste auf Basis der TF-Funktion

Die berechnete Relation ist als Parameter an den Rator weiterzureichen, damit dieser die Häufigkeiten und Gewichtungen auf eine Bewertung des Tokens abbilden kann. Die Rator-Funktionen TF (absoluter Häufigkeitsansatz) und TF-IDF zur Bewertung der Token sind bereits vorgestellt worden. Ein bewertetes Token ist ein Kandidat für einen Deskriptor. Daher ist das Resultat der Rators eine Liste von Kandidaten, deren Ordnungskriterium die Kandidatenbewertung ist. Bei gleicher Bewertung erhält der zuerst gefundene Deskriptor den Vorzug. Auf Basis der gewichteten, absoluten Häufigkeit (TF-Funktion) wird für das Beispiel aus der Tabelle 7.1 folgende Kandidatenliste in der Tabelle 7.2 berechnet.

Wie werden die beschriebenen Verarbeitungsschritte nun software-technisch umgesetzt? Es erscheint sinnvoll, den gesamten Verarbeitungsprozess zusammengefasst in einer Klasse “DescriptorFactory” bereitzustellen, deren Instanzen zur Berechnung von Such- und Filterbegriffen, sowie von Spaltenbezeichnungen der Clustermatrix und Clusterlabels verwendet werden können. Ein Objekt dieser Klasse sollte mit einer Bewertungsfunktion für Token parameterisierbar sein, um selbige austauschen zu können. In der Abbildung 7.7 ist das UML Klassendiagramm der “DescriptorFactory” angegeben.

Die Klasse “DescriptorFactory” beinhaltet den Tokenizer und den Optimizer. Zur Initialisierung benötigt ein Objekt der Klasse zusätzlich einen Rator, also eine Bewertungsfunktion für Token. Die Klassen der Bewertungsfunktionen müssen die Schnittstelle “IRatorFunction” implementieren. Die “DescriptorFactory” erhält ihre Bewertungsfunktion über ihren Konstruktor. Ein Dokument, z.B. der Inhalt einer LAssi-Karteikarte, wird dem Tokenizer in der “DescriptorFactory” als Parameter “weightedStrings” über die Methode add() gereicht. Der Wert dieses Parameters kann direkt von einem “IContent”-Objekt über die Methode “getWeightedStrings()” abgerufen werden. Der Parameter “url” der Methode “add()” wird zur Befüllung der Spalte “Dokumente” der Tokenrelation des Optimizers (siehe Tabelle 7.1) benötigt.

Die Klassen “TfRator” und “TfIdfRator”, welche die Bewertungsfunktionen für Token ent-

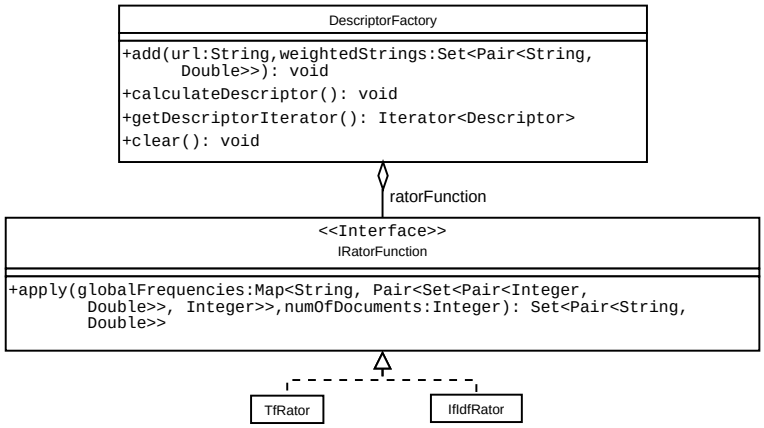

Abbildung 7.7: UML-Klassendiagramm der Klasse “DescriptorFactory’

halten, müssen die Schnittstelle “IRatorFunction” implementieren. Diese Schnittstelle gibt eine einzige Methode apply() vor, welche eine Menge von gewichteten Token, also Deskriptorkandidaten, zurückliefert. Die Instanz der Klasse “DescriptorFactory” sortiert diese Menge anschließend absteigend nach Tokenbewertung. Die vom Optimizer erzeugte Relation wird als Parameter “tokenRelation” an die “IRatorFunction” übergeben. Die Map-Datenstruktur dieses Parameters ist in Java-Notation mit Typparametern (Generics) in der Abbildung 7.8 aufgeschlüsselt worden.

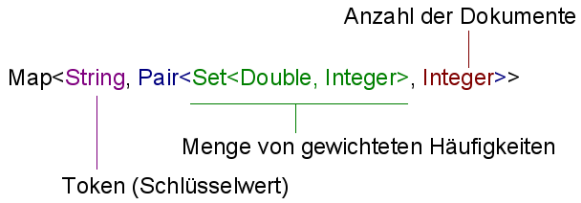

Abbildung 7.8: Datenstruktur des Parameters “tokenRelation” (“IRatorFunction”)

Über die Methode “getDescriptorIterator()” der Klasse “DescriptorFactory” kann ein Iterator abgerufen werden, mit dem die Resultatmenge der Methode “apply()” einer “IRatorFunction” sortiert nach Tokenbewertung traversiert werden kann (siehe Beispielkandidatenliste in Tabelle 7.2). Nachdem Deskriptoren aus der Kandidatenliste abgerufen worden sind, kann die Kandidatenliste und die Tokenrelation über Aufruf der Methode “clear()” gelöscht werden.

7.4 Berechnung von Anfragen

Dieses Kapitel fügt die vorgestellte Bereitstellung von gewichteten Zeichenketten (“IContentProvider”) und die Berechnung von Deskriptoren (“DescriptorFactory”) im Entwurf der Klasse “QueryService” zur Berechnung von Anfragen für die externen Suchmaschinen zusammen. Das UML-Klassendiagramm dieser Klasse ist in der Abbildung 7.9 wiedergegeben.

Die Berechnung der Suchanfragen für die integrierten Suchmaschinen ist eine Dienstleistung, die an einer zentralen Stelle im Assistenten bereitgestellt werden soll. Daher ist absehbar, dass nur eine einzige Instanz dieser Klasse “QueryService” im System existieren wird. Diese Eigenschaft soll im Entwurf durch Anwendung des Singleton-Entwurfsmusters verankert werden. Das Singleton-Entwurfsmuster beschreibt die Privatisierung des Konstruktors und die Bereitstellung des einzigen Objektes einer Klasse über eine statische Methode “getInstance()”[6]. “getInstance()” kann ohne eine Referenz auf eine Instanz der Klasse aufgerufen werden, weil sie als statische Methode an die Klasse und nicht an deren Objekte gebunden ist. Innerhalb dieser Methode wird geprüft, ob bereits eine Instanz der Klasse existiert. Sollte das der Fall sein, so liefert die Methode diese, ansonsten eine neu erzeugte Instanz, zurück (siehe UML-Listing der Methode “getInstance()”).

Der “QueryService” berechnet die Suchbegriffe auf Basis der Inhalte vom LAssi-Desktop und den hochwertigen, zu berücksichtigenden Treffern. Die Filterbegriffe werden von ihm aus den Inhalten der geringwertigen Treffer berechnet. Die Contentprovider, welche die gewichteten Zeichenketten bereitstellen, werden dem “QueryService” über die Methoden “addPositiveContentProvider()” und “addNegativeContentProvider()” bereitgestellt. Aus positiven Contentprovidern werden Such- und aus negativen Contentprovidern Filterbegriffe berechnet. Ein Objekt der Klasse “DesktopContentProvider” ist daher ein positiver Contentprovider. Instanzen der Klasse “HitContentProvider” sind immer an eine einzige Trefferschale gebunden. Daher existieren zwei Instanzen dieser Klasse im System: Eine für die hochwertigen, zu berücksichtigenden (positiver Contentprovider) und eine für die geringwertigen Treffer (negativer Contentprovider). Zur Abmeldung der Contentprovider vom “QueryService” existieren entsprechende “remove...()”-Methoden.

Die Methode “configure()” des “QueryService” erzeugt zwei “DescriptorFactory”-Objekte, welche über die Referenzen “positiveFactory” und “negativeFactory” erreicht werden können. Als “IRatorFunction” wird der Parameter “rator” an die Deskriptorfabriken gereicht. Um die Suchbegriffe bzw. die Filterbegriffe zu berechnen, muss die Methode “computePosDescriptors()” bzw. für Filterbegriffe die Methode “computeNegDescriptors()” aufgerufen werden. Wie im UML-Listing der Methode “computePosDescriptors()” angegeben ist, werden zuerst

[6] vgl. [4], Seite 157 ff.

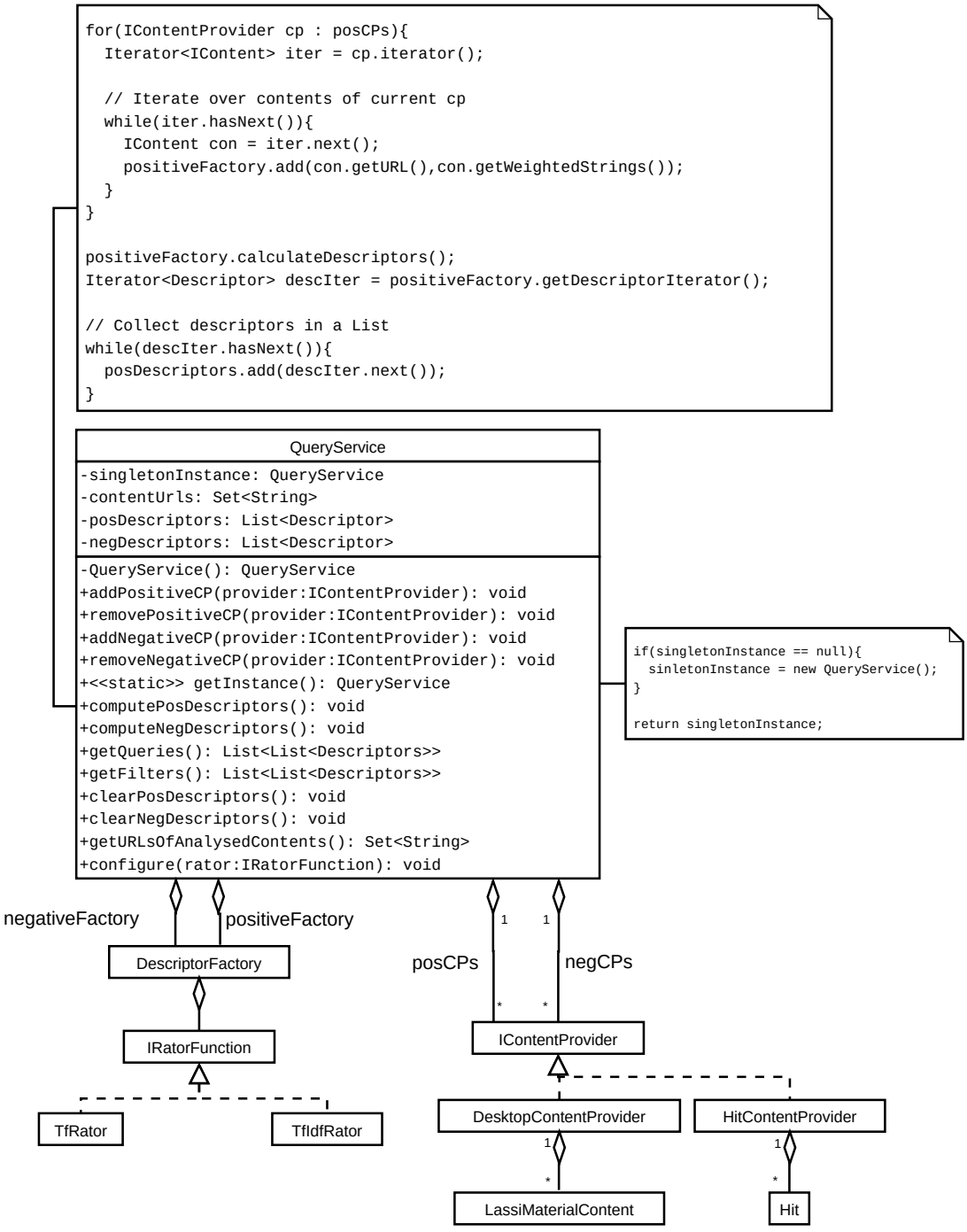

Abbildung 7.9: UML-Diagramm der Klasse “QueryService”

die Inhalte aller “IContent”-Instanzen aller Contentprovider an die zugehörige “DescriptorFactory” übergeben. Anschließend berechnet die “DescriptorFactory” die Kandidatenliste aus den bereitgestellten gewichteten Zeichenketten. Abschließend wird über ein Iterator-Objekt die Kandidatenliste traversiert und in einer eigenen Liste (Attribut “posDescriptors”) gespeichert.

Bisher ist geklärt worden, wie Such- und Filterbegriffe berechnet werden. Offen ist noch, wie aus diesen Begriffen Anfragen formuliert werden. Ein Suchbegriff entspricht einem Deskriptor, ein Filterbegriff hingegen einem negierten Deskriptor. Eine beispielhafte Anfrage A, welche das Auftreten der Deskriptoren “garnelen”, “amrum” und “lebensraum wattenmeer” und den Ausschluss des Deskriptors “rezept” realisiert, könnte in Pseudosyntax wie folgt lauten:

$$A = \text{“}garnelen\text{”}\ UND\ \text{“}amrum\text{”}\ UND\ \text{“}lebensraum\ wattenmeer\text{”}\ UND\ NICHT\ \text{“}rezept\text{”}$$

Bei der Zusammenstellung der Anfragen aus Deskriptoren und negierten Deskriptoren, verfolgt der Assistent die Strategie, die strengstmöglichen Anfrage an einen Suchdienst zu stellen, um die präzisestmögliche Trefferausbeute zu erhalten. Nun ist es aber nicht unwahrscheinlich, dass der Schüler eine höhere Anzahl von Treffern wünscht, als ein Suchdienst auf die strengstmögliche Anfrage liefert. Insofern erscheint es sinnvoll solange Anfragen an diesen Suchdienst zu richten, bis entweder die vom Schüler gewünschte Anzahl von Treffern gesammelt worden ist oder keine weiteren Anfragen mehr möglich sind. Also ist eine Liste von Anfragen zu berechnen, deren Ordnungskriterium absteigend die Strenge der Anfragen darstellt.

Grundsätzlich gilt, dass alle Anfragen in konjunktiver Normalform der Aussagenlogik zu stellen sind. Alle Deskriptoren und negierten Deskriptoren werden mit dem logischen UND-Operator verbunden. Eine solche Anfrage fordert die Existenz aller Suchbegriffe in den Treffern ein, woraus sich eine präzisere Trefferausbeute ergibt, als aus einer Anfrage, die nur mindestens einen Begriff in ihren Treffern verlangt. Eine Anfrage A ist genau dann strenger als eine Anfrage B, wenn

1. die Kardinalität von A (die Anzahl der enthaltenen Deskriptoren) größer ist, als die Kardinalität von B

2. bei gleicher Kardinalität, wenn die mittlere arithmetische Deskriptorbewertung durch den Rator der “DescriptorFactory” von A größer ist, als die mittlere arithmetische Deskriptorbewertung von B

Bei gleicher Kardinalität und gleichem mittleren Gewicht gelten zwei Anfragen als gleich

streng. Bei der Berechnung der Kardinalität einer Anfrage sind Deskriptor und negierter Deskriptor gleichwertig, da beide Deskriptoren Beschränkungen der Ergebnismenge definieren. Formal notiert lautet die Ordnungsdefinition auf Basis der Strenge einer Anfrage:

$$strict(A) > strict(B) \Leftrightarrow (card(A) > card(B)) \vee ((card(A) = card(B)) \wedge (avgWeight(A) > avgWeight(B))) \tag{7.2}$$

$$strict(A) = strict(B) \Leftrightarrow card(A) = card(B)) \wedge (avgWeight(A) = avgWeight(B) \tag{7.3}$$

$$strict(A) < strict(B) sonst \tag{7.4}$$

Ein einfaches Verfahren zur Berechnung der nach Strenge sortierten Liste der Anfragen lautet:

1. Berechne alle Kombinationen aller Deskriptoren (jede Kombination repräsentiert eine Anfrage).
2. Sortiere die in Schritt eins erhaltene Menge nach Strenge der Anfragen

Die Berechnung aller Kombinationen aller Deskriptoren entspricht der Berechnung der Potenzmenge, also der Menge aller Teilmengen, ohne die leere Menge über die Liste “posDescriptors” für Such- bzw. “begDescriptors” für Filterbegriffe. Über die Methoden “getQueries()” bzw. “getFilters()” können die jeweiligen Potenzmengen berechnet und vom “Query-Service” abgerufen werden.

Die formulierten Suchanfragen enthalten immer zuerst alle Such- und anschließend alle Filterbegriffe. Anfragen, die nur Filterbegriffe enthalten sind unsinnig und werden daher nicht gestellt. Für das obige Beispiel sind die berechneten Suchanfragen inklusive Filterbegriffen in der Tabelle 7.3 angegeben. Die Bewertungen des Rators lauten:

- “garnelen”: 6.0 Punkte (Suchbegriff)
- “amrum”: 4.0 Punkte (Suchbegriff)
- “lebensraum wattenmeer”: 6.0 Punkte (Suchbegriff)
- “rezept” : 4.0 Punkte (Filterbegriff)

Dieses Verfahren berücksichtigt allerdings nicht die Elemenierung der Deskriptorredundanz innerhalb einer Anfrage. Eine Anfrage enthält genau dann einen redundanten Deskriptor, wenn dieser prinzipiell nicht zu einer Präzisierung der Trefferausbeute führen kann. Das

Rang	Anfrage
1.	“garnelen” UND “amrum” UND “lebensraum wattenmeer” UND NICHT “rezept”
2.	“garnelen” UND “lebensraum wattenmeer” UND NICHT “rezept”
3.	“garnelen” UND “amrum” UND NICHT “rezept”
4.	“lebensraum wattenmeer” UND “amrum” UND NICHT “rezept”
5.	“garnelen” UND NICHT “rezept”
6.	“lebensraum wattenmeer” UND NICHT “rezept”
7.	“amrum” UND NICHT “rezept”
8.	“garnelen” UND “amrum” UND “lebensraum wattenmeer”
9.	“garnelen” UND “lebensraum wattenmeer”
10.	“garnelen” UND “amrum”
11.	“lebensraum wattenmeer” UND “amrum”
12.	“garnelen”
13.	“lebensraum wattenmeer”
14.	“amrum”

Tabelle 7.3: Anfragen aus Such- und Filterbegriffen zum Garnelen-Beispiel

Problem der Deskriptorredundanz soll anhand des folgenden Beispiels der Anfrage A veranschaulicht werden:

$$A = \text{“}lebensraum\text{”}\ UND\ \text{“}wattenmeer\text{”}\ UND\ \text{“}lebensraum\ wattenmeer\text{”}$$

Die Anfrage A beinhaltet zwei Deskriptoren “lebensraum” und “wattenmeer”, die nicht zu einer signifikanten Einschränkung der Ergebnismenge führen. Es werden nämlich nur Treffer geliefert, in denen beide Suchbegriffe direkt hintereinander vorkommen (spezifiziert durch Deskriptor “lebensraum wattenmeer”). Daher könnten die Suchbegriffe “lebensraum” UND “wattenmeer” durch andere ersetzt werden, z.B. durch “schrimp“ oder “garnelenzucht”. Ein Verfahren zur Lösung dieses Problems ist im Entwurf des Assistentensystems bisher noch nicht vorgesehen und bietet daher Raum für weitere Entwicklungen. Da eine Anfrage eine Liste von Such- bzw. Filterbriffen ist (Ordnungskriterium: Deskriptorbewertung) und auch die Menge der Anfragen eine Ordnung hat (Ordnungskriterien: Kardinalität und mittleres Deskriptorgewicht einer Anfrage), ist das Resultat der Methoden “getQueries()” bzw. “getFilters()” eine Liste von Listen von Deskriptoren.

Um die Tokenrelation der Deskriptorfabriken des “QueryService” zu leeren, existieren zwei Methoden: “clearPosDescriptors()” und “clearNegDescriptors()”. Über die Methode “getURLsOfAnalysedContents()” kann eine Menge von URLs aller bereitgestellten “IContent”-Instanzen der Contentprovider abgerufen werden. Diese Menge ist notwendig, damit bei der Trefferrecherche in den Suchmaschinen bereits gefundene Treffer nicht erneut gefunden werden.

7.5 Interaktion mit integrierten Suchmaschinen

7.5.1 Repräsentation einer Suchmaschine und eines Treffers

Um die Interaktion des Assistenten mit den integrierten Suchmaschinen zu klären, muss zuerst die Repräsentation einer Suchmaschine und ihrer Treffer im Assistenten entworfen werden. Interaktion meint in diesem Zusammenhang konkret das Senden von Suchanfragen an die Suchmaschine und das Empfangen, sowie das Verwalten ihrer Treffer.

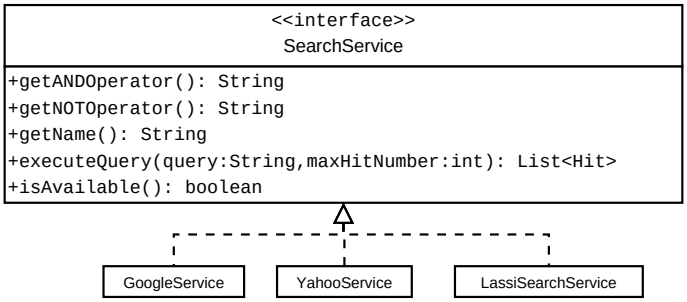

Abbildung 7.10: UML-Diagramm der Schnittstelle “SearchService”

Eine Suchmaschine wird durch die Schnittstelle “SearchService” repräsentiert, deren Entwurf in UML-Notation in der Abbildung 7.10 dargestellt ist. Die Syntax, in der Suchanfragen formuliert werden, unterscheidet sich von Suchmaschine zu Suchmaschine (Beispiel: “UND“, “AND” oder “+” für das logische UND). Daher muss eine Suchmaschine ihre Notation für den UND- und den NICHT-Operator bereitstellen. In der Schnittstelle “SearchService” stehen hierfür die Methoden “getANDOperator()” und “getNOTOperator()” zur Verfügung. Für die Herkunftsansicht wird außerdem eine Bezeichnung der Suchmaschine benötigt, welche die Methode “getName()” liefert. Die Methode “getID()” liefert eine eindeutig identifizierende Zeichenkette für den Suchdienst. Da die ID auf diese Weise vom Entwickler des Suchmaschinenadapters festgelegt werden muss, besteht die Möglichkeit, dass zwei Suchdienste dieselbe ID verwenden. Um dieses Problem zu entschärfen ist in der Dokumentation der Methode die Konvention festgelegt worden, dass die ID aus dem vollständigen Namen des Klassenpaketes und dem Namen der Suchmaschine bestehen soll. Als Beispiel sei hier die ID “org.lassitools.searchassistent.searchservice.Yahoo” für die Suchmaschine Yahoo genannt.

Eine Suchanfrage wird in Form eines Strings als Parameter “query” der Methode “executeQuery()” an die adaptierte Suchmaschine gesendet. Zusätzlich wird der Suchmaschine die maximale Anzahl von Treffern über den Parameter “maxHitNumber” mitgeteilt, die diese zur Anfrage “query” liefern darf. Es gilt hier die Invariante, dass die Länge der Resultatliste von Treffern, die diese Methode liefert, nicht länger als “maxHitNumber” Elemente sein darf. Die

Anzahl der zu liefernden Treffer wird vom Schüler spezifiziert. Daher ist der Wert von “maxHitNumber” bei der ersten Suchanfrage gleich der Spezifikation des Schülers. Sollte die erste Anfrage weniger Treffer liefern, als der Schüler wünscht, so wird “maxHitNumber” in den folgenden Anfragen jeweils die Anzahl der fehlenden Treffer zugewiesen. Vor jedem Aufruf von “executeQuery()” wird die Methode “isAvailable()” ausgeführt, welche testet, ob mit der Suchmaschine kommuniziert werden kann. Dies ist notwendig, um die Erreichbarkeit entfernter Suchmaschinen zu überprüfen und im Negativfall eine Fehlermeldung zu erzeugen. Die definierte Schnittstelle “SearchService” muss von den Adapterklassen der Suchmaschinen, z.B. “YahooService” oder “LassiSearchService” implementiert werden.

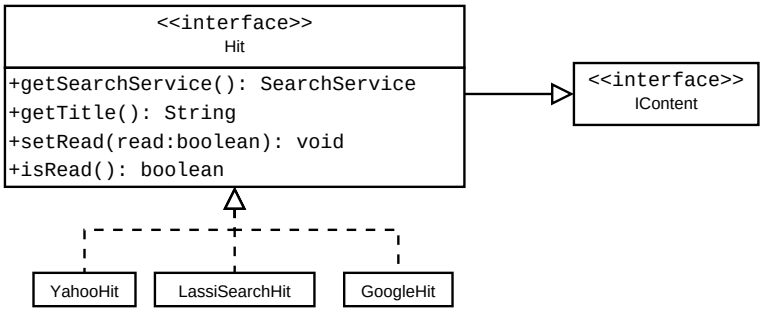

Abbildung 7.11: UML-Diagramm der Klasse “Hit”

Die Treffer, welche die Methode “executeQuery()” der Schnittstelle “SearchService” liefert, werden durch Implementierungen der Schnittstelle “Hit” abgebildet. Die UML-Notation eines “Hit” zeigt die Abbildung 7.11. Da sowohl hochwertige, zu berücksichtigende, als auch geringwertige Treffer zur Berechnung neuer Such- und Filterbegriffe genutzt werden, erbt die Schnittstelle “Hit” von der Schnittstelle “IContent” und gibt damit deren Implementierung direkt vor. Um die Herkunftsansicht zu berechnen, muss von einem Treffer die Suchmaschine abgerufen werden können, die den Treffer gefunden hat. Dies ist über die Methode “getSearchService()” möglich. Außerdem muss jeder Treffer einen aussagekräftigen Titel über die Methode “getTitle()” bereitstellen, welcher sowohl in der Herkunfts- als auch in der Clusteransicht angezeigt werden kann. Da die Treffer anhand ihres URL identifiziert werden sollen, um die Anzahl von Doubletten zu vermindern, muss der URL eines jeden Treffers abrufbar sein. Dies geschieht über die Methode “getURL()” (geerbt von “IContent”). Über die Methoden “isRead()” und “setRead()” kann der “Gelesen-Zustand” eines Treffers abgerufen bzw. gesetzt werden. Dieser Zustand dient zur optischen Hervorhebung von bereits geöffneten Treffern in den Trefferansichten. Die gewichteten Inhalte eines “Hit” können über die geerbte Methode “getWeightedStrings()” abgerufen werden.

7.5.2 Kommunikation mit Suchmaschinen

Offen geblieben ist bisher, wer die Anfragen an die Suchmaschinen sendet und deren Treffer entgegennimmt. Es gibt dazu prinzipiell zwei Möglichkeiten: entweder werden alle Suchmaschinen nacheinander oder nebenläufig befragt. Die nebenläufige Variante hat den Vorteil, dass Latenzzeiten, z.B. von entfernten Suchmaschinen, für das Senden von weiteren Anfragen an andere Suchmaschinen genutzt werden können. Auf diese Weise kann der Recherchedurchgang des Assistenten beschleunigt werden. Daher wird das nebenläufige Modell im Entwurf umgesetzt.

An dieser Stelle ist das Konzept eines Reflex-Agenten, wie es Russel und Norvig beschreiben, sinnvoll einsetzbar[7]. Nach Russel und Norvig ist jedes Lebewesen oder jedes Objekt ein Agent, das seine Umwelt über Aktuatoren beeinflusst und über Sensoren wahrnimmt[8]. Ein Reflex-Agent ist die einfachste Form eines Agenten. Er handelt nur aufgrund seiner aktuellen Wahrnehmung und ist nicht in der Lage, einen Wahrnehmungsverlauf zu speichern.

Es könnte für jede Suchmaschine ein Reflex-Suchagent bereitgestellt werden. Die Umwelt eines solchen Agenten ist die Suchmaschine, in der er recherchieren soll. Er handelt durch das Übermitteln einer Anfrage an die Suchmaschine und nimmt deren Treffer entgegen, was der Wahrnehmung seiner Umwelt entspricht. Die Anfrage stammt aus der vom "QueryService" berechneten Anfragenliste, die dem Suchagenten übergeben werden muss. Nach jedem Recherchedurchgang prüft der Suchagent, ob seine Treffer von anderen Suchagenten bereits erzielt worden sind. Falls nicht, so werden sie von ihm in die Sammelschale gelegt.

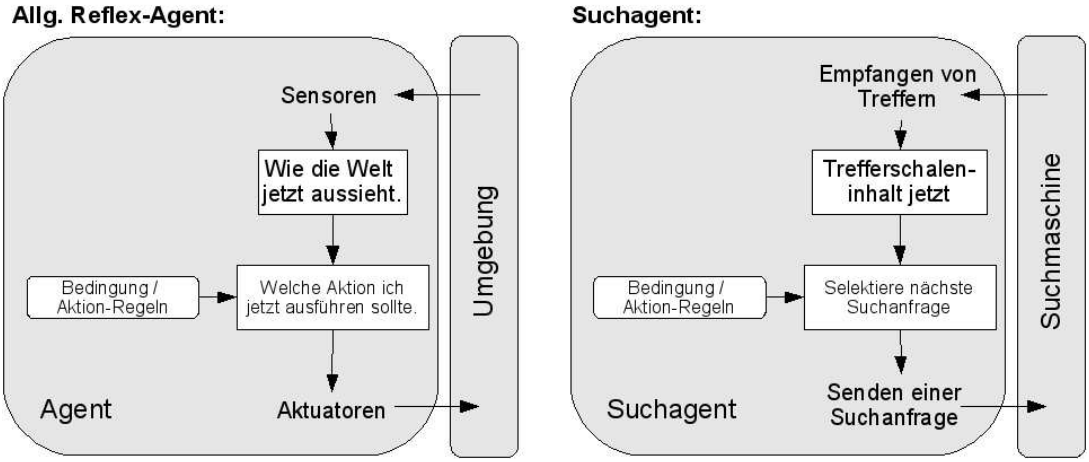

Abbildung 7.12: Allgemeiner Reflex-Agent und Suchagent

[7] vgl. [11], Seite 72 ff.
[8] vgl. [11], Seite 55 ff.

Wenn der Suchagent die vom Schüler gewünschte Trefferanzahl in die Sammelschale gelegt hat oder keine weiteren Anfragen in seiner Liste stehen, kann er seine Recherche abbrechen. Ansonsten muss er die nächste Anfrage an die Suchmaschine senden. Aufgrund des statischen Ablaufs des Rechercheprozesses ist ein Suchagent als Reflex-Agent zu klassifizieren, denn er nimmt schlicht die nächste Anfrage und wählt diese nicht auf Basis eines Erfahrungswertes aus, z.B. “Anfragen mit mehr als vier Filterbegriffen erzielen nie Treffer, die vom Schüler als “hochwertig” bewertet werden”. Das allgemeine Konzept eines Reflex-Agenten und das Konzept des Suchagenten sind in der Abbildung 7.12 gegenübergestellt.

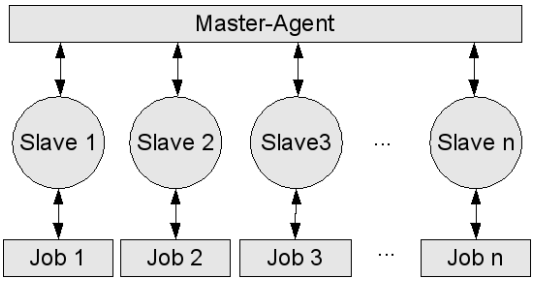

Abbildung 7.13: Schema einer Master-Slave-Architektur

Für die Organisation der Suchagenten ist der Einsatz einer Master-Slave-Architektur sinnvoll (siehe Abbildung 7.13). Es gibt einen zentralen Master-Agenten, der die Suchagenten (Slaves) nach einer bestimmten Zeitdauer erzeugt, ihnen eine Suchmaschine zuordnet und sie startet. Anschließend wartet er, bis alle Suchagenten ihre Recherchen (also ihre Jobs) abgeschlossen haben und terminiert sind. Jetzt berechnet er den Zeitpunkt des nächsten Durchgangs. Wenn der berechnete Zeitpunkt erreicht ist, beginnt er mit der erneuten Erzeugung der Suchagenten. Schaltet der Schüler das gesamte System ab, so ist es die Aufgabe des Master-Agenten, den einzelnen Suchagenten ein Signal zum Beenden der Suche zu senden.

Die software-technische Umsetzung der Interaktion des Assistenten mit den Suchmaschinen ist in UML-Notation in der Abbildung 7.14 angegeben. Die Funktion des Master-Agenten wird durch das Singleton-Objekt der Klasse “AgentManager” ausgeübt. An ihm ist es möglich, Suchmaschinen und Instanzen der Schnittstelle “IResultListener” über entsprechende “add..()”- und “remove...()”-Methoden an- bzw. abzumelden. Ein “IResultListener” ist z.B. ein Objekt, welches eine Bildschirmmeldung nach Abschluss eines Recherchedurchgangs ausgibt.

Der “AgentManager” startet den ersten Recherchedurchlauf nach Aufruf der Methode “start()”. Zuerst müssen die Anfragen im “QueryService” berechnet und dann vom “AgentManager” abgerufen werden. Anschließend erzeugt der “AgentManager” die Suchagenten. Da die Suchagenten auf Basis verschiedener Technologien implemementiert werden können, ist im Entwurf eine Schnittstelle “IAgent” vorgesehen, welche die konkrete Agententechnologie lo-

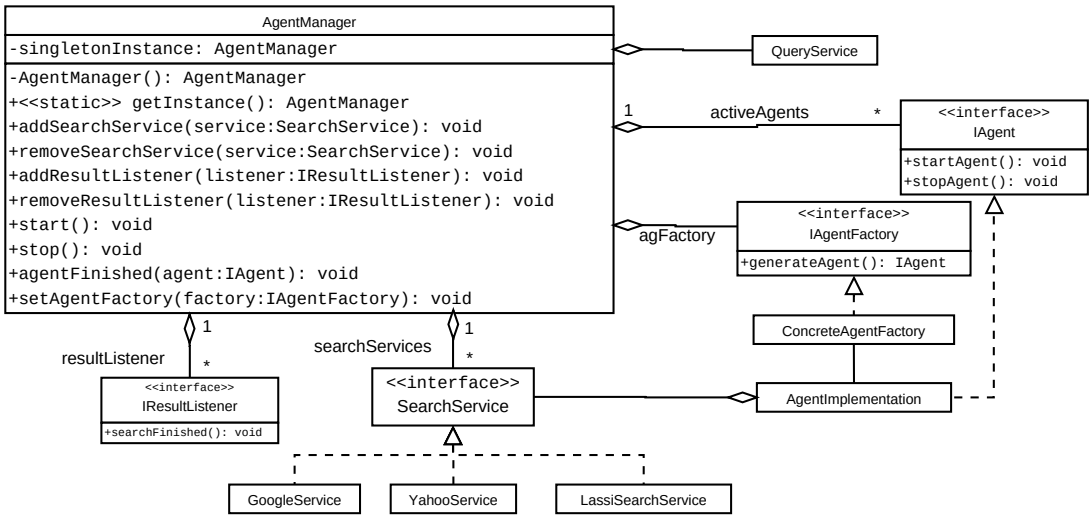

Abbildung 7.14: UML-Diagramm der Klasse “AgentManager” und ihres Kontextes

se an den “AgentManager” koppelt und damit austauschbar macht. Im UML-Diagramm ist die Suchagentenklasse allgemein als “AgentImplementation” angegeben. Jedes Objekt dieser Klasse besitzt eine Referenz auf eine, vom “AgentManager” zugeordnete, Instanz der Klasse “SearchService”. Von dieser Instanz werden die String-Repräsentationen für die aussagenlogischen Operatoren über die Methode “getANDOperator()” und “getNOTOperator()” abgerufen.

Aber wie kann der “AgentManager” die Klasse “AgentImplementation” instanziieren, die er gar nicht kennt? Zur Lösung dieses Problems ist das Entwurfsmuster einer abstrakten Fabrik verwendet worden[9]. Eine abstrakte Fabrik wird durch eine Schnittstelle definiert, welche die Implementierung einer Methode zur Erzeugung eines Produktobjektes der Fabrik vorschreibt. In einer konkreten Fabrikklasse, welche die Schnittstelle implementiert, ist dann der Konstruktoraufruf der Produktklasse hinterlegt und das neu erzeugte Objekt als Resultat zurückgeliefert. Auf diese Weise können Objekte erzeugt werden, ohne dass deren konkrete Klasse bekannt ist. Der “AgentManager” nutzt die Methode “generateAgent()” der Fabrikschnittstelle “IAgentFactory” zur Erzeugung der Suchagenten. Die Klasse “ConcreteAgentFactory” implementiert diese Methode und erzeugt dort neue Instanzen der Klasse “AgentImplementation”, welche sie als Resultat an den “AgentManager” zurückliefert. Ein Objekt der Fabrikklasse “IAgentFactory” wird dem “AgentManager” zu dessen Initialisierungszeitpunkt über die Methode “setAgentFactory()” gereicht.

Die Referenzen auf die Suchagenten werden in der Menge “activeAgents” des “AgentMa-

[9] vgl. [4], Seite 107 ff.

nagers" gespeichert. Über die Referenzen aus dieser Menge kann der "AgentManager" den einzelnen "IAgent"-Instanzen über Aufruf deren Methode "stopAgent()" ein Terminierungssignal senden. Die Methode "startAgent()" veranlasst den Suchagenten zum Absenden seiner ersten Anfrage an die ihm zugeordnete Suchmaschine. Bevor ein Suchagent terminiert, ruft er die Methode "agentFinished()" der Singleton-Instanz des "AgentManagers" auf und übergibt sich selbst als Parameter "agent". Dieses Verhalten ist als Konvention in der Methodendokumentation hinterlegt. Es kann software-technisch im aktuellen Entwurf nicht garantiert werden, dass sich die konkreten Implementierungen an diese Konvention halten. Nachdem alle Suchangenten terminiert sind, ruft der "AgentManager" die Methode "searchFinished()" aller angemeldeten Instanzen der Schnittstelle "IResultListener" auf. Auf diese Weise kann die Ausgabe der angesprochenen Bildschirmmeldung angestoßen werden. Die Methode "stop()" des "AgentManager" fährt das Master-Slave-System herunter, was konkret bedeutet, dass der "AgentManager" ein Terminierungssignal an alle Suchagenten sendet und keinen Zeitpunkt für einen weiteren Durchgang berechnet. Sollten keine Agenten aktiv sein, so wird der berechnete Zeitpunkt des nächsten Durchlaufes gelöscht.

7.5.3 Verwaltung der Treffer in Trefferschalen

Der bisherige Entwurf umfasst lediglich die Recherche von Treffern. Es sollen aber, wie bereits beschrieben, vier Schalen existieren, in denen Treffer abgelegt werden können:

- Schale für hochwertige, zu berücksichtigende Treffer
- Ablageschale für hochwertige Treffer
- Sammelschale
- Schale für geringwertige Treffer

Die Treffer sollen von den Suchagenten in der Sammelschale gespeichert und vom Schüler zwischen den Schalen hin und her verschoben werden können. Zur Verwaltung der Treffer wird die Klasse "HitService" definiert. Da die Treffer ebenfalls an einer zentralen Stelle verwaltet werden sollen, wird auch hier das Singleton-Entwurfsmuster angewandt. Die Klassenspezifikation in UML-Notation ist in der Abbildung 7.15 angegeben.

Die Klasse stellt die Methoden "addHit()", "moveHit()" und "deleteHit()" zum Hinzufügen zu einer, Verschieben in eine andere und Löschen aus einer Schale bereit. Die "basin"-Parameter identifizieren die entsprechende Trefferschale. Über die Methode "setHitBasinSize()" kann die Anzahl der Treffer, die die Schale "basin" aufnehmen kann, auf "size" Treffer begrenzt werden. Gefüllt wird die Sammelschale direkt von den Suchagenten.

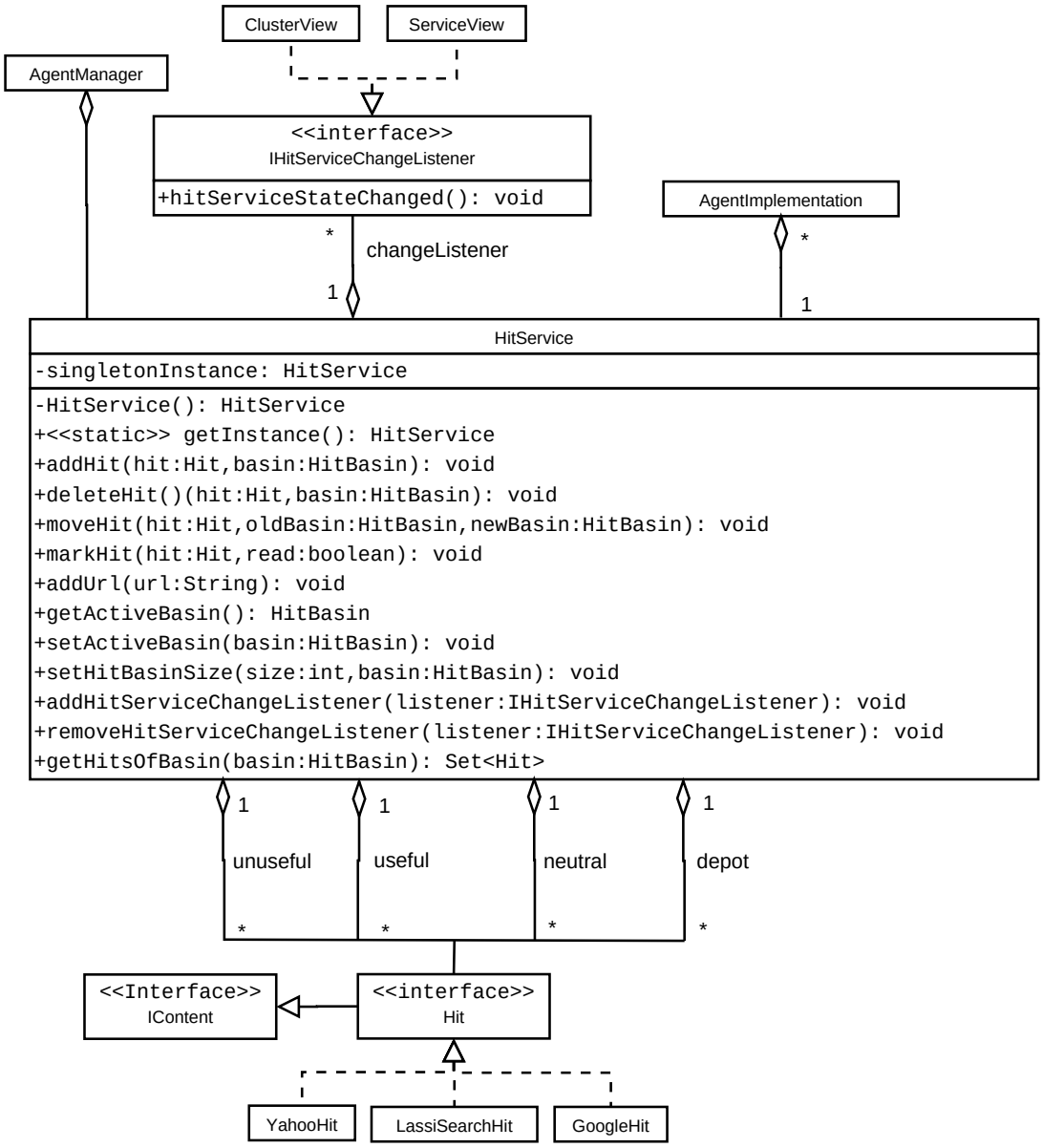

Abbildung 7.15: UML-Diagramm der Klasse “HitService” und ihres Kontextes

Die vier Trefferschalen werden wie folgt durch vier Mengen von Hit-Objekten repräsentiert:

- Hochwertige und zu berücksichtigende Treffer = “usefull”
- Hochwertige Treffer (Ablageschale) = “depot”
- Neutrale Treffer (Sammelschale) = ”neutral”
- Geringwertige Treffer (Abfallschale) = “unuseful”

Die Klassen von Objekten, die über Veränderungen an den Trefferschalen informiert werden sollen, müssen die Schnittstelle “HitServiceChangeListener” implementieren und sich über die Methode “addHitServiceChangeListener()” am “HitService” anmelden. Sobald eine Veränderung an den Trefferschalen vorgenommen wird, z.B. durch Hinzufügen eines neuen Treffers durch einen Suchagenten, wird die Methode “hitServiceStateChanged()” aller angemeldeten “HitServiceChangeListener”-Instanzen vom “HitService” aufgerufen. Beispiele für “HitServiceChangeListener”-Instanzen sind die zwei Trefferansichten. Sie rufen nach einer Zustandsänderung einer Schale die aktuell vom Schüler selektierte Schale ab, bereiten diese auf und stellen sie dar.

Die Singleton-Instanz der Klasse “HitService” garantiert, dass ein Treffer mit einer URL nicht mehrfach gefunden wird. Die einzelnen Suchagenten rufen die Methode “addHit()” zum Speichern eines Treffers auf. Da die Suchagenten nebenläufig arbeiten, müssen alle Methodenaufrufe auf dem “HitService” zur Konsistenzgarantie synchronisiert erfolgen. Es muss also sichergestellt werden, dass zwei Agenten nicht gleichzeitig eine Methode des “HitService” aufrufen können. Ist ein Treffer mit einem bestimmten URL bereits in einer Schale enthalten, so muss eine “IsAlreadyIn”-Ausnahme geworfen werden. Der jeweilige Suchagent erkennt anhand dieser Ausnahme, dass ein anderer Suchagent bereits diesen Treffer recherchiert hat. Daher muss er die Recherche fortsetzen, um einen weiteren Treffer zu erzielen, den er in den “HitService” einfügen kann. Eine Alternative zu diesem Verfahren wäre die Kommunikation der Suchagenten untereinander. Ein Suchagent müsste alle anderen befragen, ob sie bereits einen Treffer mit einem bestimmten URL recherchiert haben. Wenn dies nicht der Fall ist, dann kann der Suchagent seinen Treffer in den “HitService” einfügen. Von dieser Variante wird abgesehen, weil die Implementierung aufwendiger als die vom ersten Verfahren wäre und sie keinen Mehrwert bietet, der im aktuellen Entwurf genutzt werden kann.

7.6 Entwurf der Clusterberechnung

Die Clusterberechnung ist Voraussetzung für die Clusteransicht, die der Assistent dem Schüler anbieten soll. Die Berechnung der Cluster in Abhängigkeit zum Trefferinhalt muss vom Assistenten selbst ausgeführt werden, da die Treffer aller Suchmaschinen in der Clusteransicht dargestellt werden sollen und die meisten Suchmaschinen keine Treffercluster zum Abruf bereitstellen.

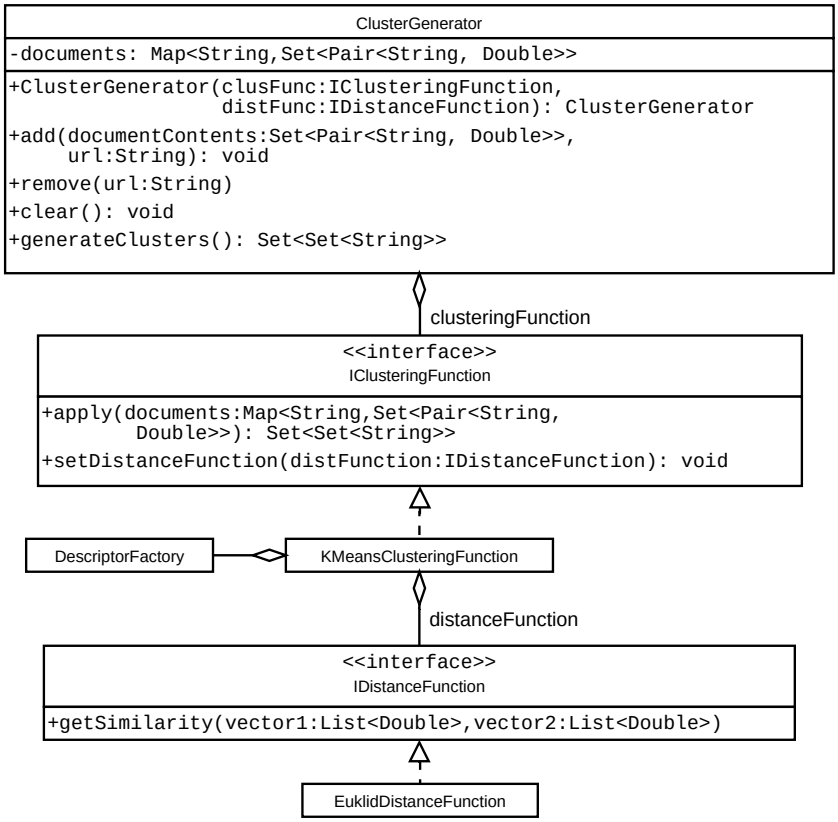

Abbildung 7.16: UML-Diagramm der Klasse “ClusterGenerator” und ihres Kontextes

Wie in UML-Notation in der Abbildung 7.16 dargestellt, soll die Clusterberechnung über ein Objekt der Klasse “ClusterGenerator” abgewickelt werden. Der Konstruktor der Klasse “ClusterGenerator” erwartet als Parameter ein Clustering-Verfahren, z.B. das k-Means-Verfahren, und eine Distanzfunktion für das Verfahren, z.B. die euklidische Distanz. Die Klasse, die das Clustering-Verfahren implementiert, muss die Schnittstelle “IClusteringFunction” bedienen. Die zu implementierende Schnittstelle für die Distanzfunktion heißt “IDistanceFunction”. Ein Objekt der “IDistanceFunction” wird direkt über die Methode “setDistanceFunction()” an ein Objekt der “IClusteringFunction” weitergereicht. Aufgrund der lo-

sen Kopplung des Clustering-Verfahrens an den “ClusterGenerator” und der Distanzfunktion an das Clustering-Verfahren, können beide gegen Alternativen ausgetauscht werden. Über die Methode “add()” können dem “ClusterGenerator” Trefferinhalte übergeben werden. Sie erwartet gewichtete Strings als Dokumentinhalt und den URL des Dokuments. Der “ClusterGenerator” sammelt die Dokumentinhalte in der Map “documents”, deren Schlüssel der Dokument-URL und deren Wert die Menge von gewichteten Textinhalten darstellt. Falls ein bereits hinzugefügtes Dokument wieder entfernt werden soll, ist die Methode “remove()” zu verwenden, welche die Inhalte zur übergebenen URL aus der Map “documents” löscht. Die Methode “clear()” löscht die gesamte Map. Nach Aufruf der Methode “generateClusters()” wird die Map über die Methode “apply()” an das Objekt der “IClusteringFunction” implementierenden Klasse gereicht. Diese Methode gibt die gewichteten Strings der Map direkt in eine erzeugte Instanz der Klasse “DescriptorFactory”, um die Spaltenbezeichnung der Merkmalsmatrix X zu erhalten. Der URL wird als Dokumentidentifikator verwendet, denn das Resultat des Clustering-Prozesses ist eine Menge von Mengen von URLs. Durch diese Abstraktion von den “Hit”-Objekten, welche in der Clusteransicht dargestellt werden sollen, ist ein Objekt der Klasse “ClusterGenerator” in der Lage, beliebige andere Dokumente zu clustern. Diese Klasse kann daher unkompliziert wiederverwendet werden. Ein Objekt, welches eine Instanz des “ClusterGenerator” nutzt, muss eine Abbildung vom URL auf die eigentlichen Dokumentobjekte besitzen, damit die abstrakten URL-Cluster auf konkrete Dokumentobjektcluster abgebildet werden können. Ein Objekt der Klasse für die Clusteransicht wird demnach eine Map von URL (String) auf “Hit”-Objekt nutzen, um Treffercluster darstellen zu können.

Die Cluster werden von der Methode “generateClusters()” berechnet, welche eine Menge von Clustern zurückliefert. In den Cluster-Mengen sind die Treffer-URLs enthalten, welche später auf die eigentlichen Trefferobjekte abgebildet werden müssen. Die Methode “generateClusters()” deligiert die Clusterberechnung an die “apply()”-Methode der Schnittstelle “IClusteringFunction” und übergibt dieser die Menge “documents” und liefert deren Resultmenge von URL-Mengen zurück.

Aufgrund der linearen Laufzeitkomplexität und der guten Ergebnisse, von denen die Literatur berichtet, soll das k-Means-Verfahren im Entwurf als “IClusteringFunction” verwendet werden. Um die Spalten der Merkmalsmatrix X zu beschriften, müssen Deskriptoren berechnet werden, deren gewichtete Häufigkeiten dann pro Treffer in die zugehörigen Zellen eingetragen werden. Hierfür soll die k-Means-Implementierung des Assistenten ein Objekt der bereits entworfenen Klasse “DeskriptorFactory” nutzen. Als Distanzfunktion “IDistanceFunction” ist im Entwurf die euklidische Distanz vorgesehen.

8 Implementierungsdetails

Dieses Kapitel beschreibt ausgewählte Details der Implementierung des Assistenten. Grundsätzlich sei gesagt, dass das System in der Programmiersprache Java auf Basis des Frameworks Eclipse RCP (Rich Client Platform) entwickelt worden ist, weil die LAssi-Plattform ebenfalls auf diesen Technologien basiert. Die Eclipse-Plattform bietet ein komfortables und ausgereiftes Plugin-Konzept. Da der Assistent in Form eines Eclipse-Plugins entwickelt worden ist, kann er über dieses Konzept nahtlos in die LAssi-Plattform integriert werden.

8.1 Extraktion von Trefferinhalten

Der Assistent benötigt sowohl für das Finden von Such- und Filterbegriffen, als auch für das Berechnen von Trefferclustern die Textinhalte der Treffer. Diese nutzen aber hochgradig heterogene Datenformate, z.B. indexiert die Web-Suchmaschine Yahoo u.a. HTML-, PDF-, RTF-, und Word-Dokumente. Wie ist die Extraktion der Daten aus diesen Treffernformaten über die Methode “getContents()” der Schnittstelle “Hit” in ihren konkreten Klassen (z.B. “YahooHit”) implementiert?

Als Beispiel wird hier die Implementierung von Treffern der Suchmaschine Yahoo betrachtet (Klasse “YahooHit” in der Abbildung 8.1). Die LAssi-Plattform stellt ein Singleton-Objekt der Klasse “DocumentReaderService” bereit. Die Aufgabe dieses Dienstes ist das Konvertieren eines Dokuments in eine Menge von gewichteten Zeichenketten. Diese Funktionalität wird über die Methode “readDocumentsContents()” abgebildet. Die Methode erwartet die Angabe des Dokument-MIME-Typs und eines Objektes der Klasse “java.io.InputStream“. Ein “InputStream“-Objekt beinhaltet eine Referenz auf eine Ressource und bietet über die Methode “read()” die Möglichkeit, diese Ressource byteweise auszulesen. MIME steht für Multipurpose Internet Mail Extensions und ist ein Kodierstandard, welcher u.a. den Typ eines, über ein Netzwerk zu übertragenden, Dokumentes in Form einer Zeichenkette festlegt. Anhand des übergebenen MIME-Typs wird der Konvertor für das Trefferdokument vom “DocumentReaderService” über die Map “readers” ausgewählt. Diese Map ordnet einem MIME-Typ ein Objekt der Schnittstelle “IDocumentReader” zu. Diese Schnittstelle muss von den Konvertoren implementiert werden, welche vom Entwickler der Suchmaschinen-Integration über einen

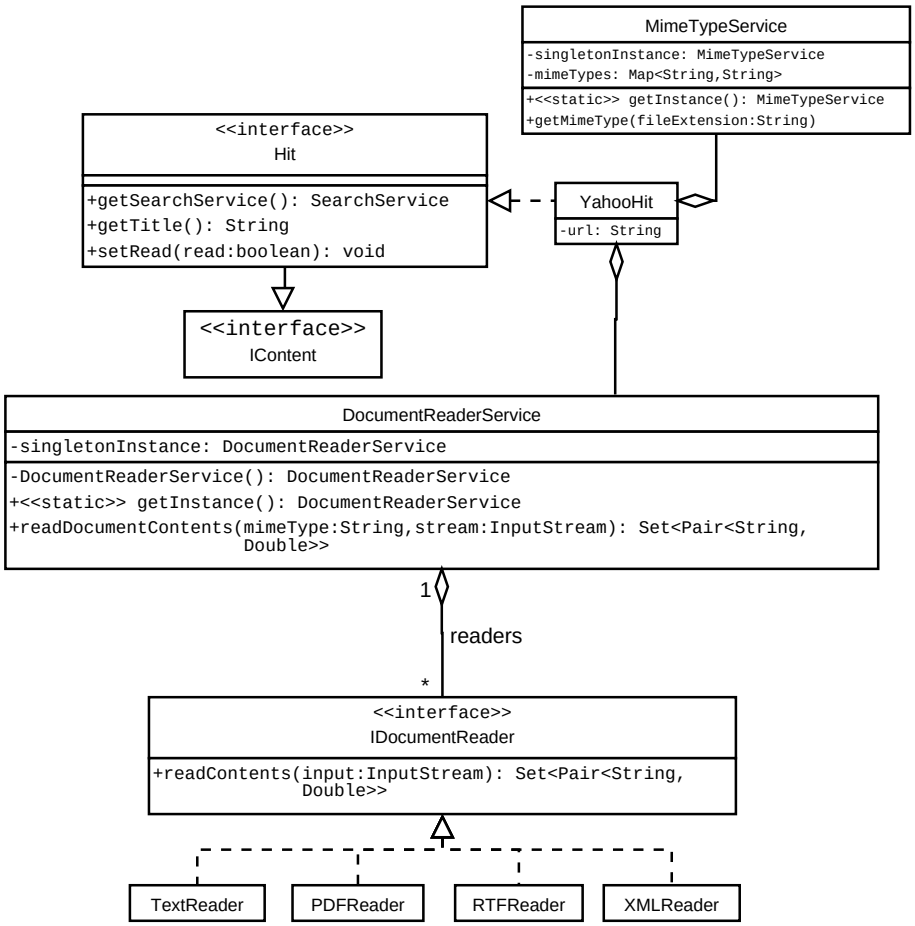

Abbildung 8.1: UML-Klassendiagramm der Klasse “DocumentReaderService”

Erweiterungspunkt des “DocumentReaderService” bereitzustellen sind.

Ein Yahoo-Treffer speichert seinen URL. Aus diesem wird nach Aufruf der Methode “getContents()” die Dateiendung der Trefferdatei extrahiert und an die Singleton-Instanz der Klasse “MimeTypeService” gereicht. Dieser Dienst ordnet über eine Map der Dateiendung die standardisierte Bezeichnung des MIME-Typs dieser Trefferdatei zu. Die standardisierten Bezeichnungen stammen aus einer Textdatei, welche zum Erzeugungszeitpunkt des “MimeTypeService” von dessen Konstruktor ausgelesen wird. Die Erzeugung eines “java.io.InputStream”-Objektes wird an eine Instanz der Klasse “java.net.URL” delegiert. Diese Klasse erwartet im Konstruktor die Angabe eines URL als String, welcher jedem Yahoo-Treffer bekannt ist. Das erzeugte “java.net.URL”-Objekt bietet eine Operation “openStream()”, welche eine, mit dem Trefferdokument assoziierte, Instanz von “java.io.InputStream” liefert. Dieses Objekt kann dann mitsamt des ermittelten MIME-Typs an die Methode “readDocumentContents()” des

“DocumentReaderService” gereicht und deren Resultat zurückgegeben werden.

Diese Implementierung erleichtert Entwicklern neuer Suchmaschinen-Anbindungen das Integrieren ihrer Trefferformate, weil sie lediglich dann einen Konvertor bereitstellen müssen, wenn noch keiner für ihr Trefferformat existiert. Die Dienstleistung aller im System existierenden Konvertoren ist über den zentralen Dienst “DocumentReaderService” komfortabel zugreifbar.

8.2 Statusmonitoring der Suchagenten

Der Schüler soll über den Fortschritt des Suchprozesses der Suchagenten informiert werden, damit er bei umfangreichen Recherchen der Agenten nicht an einen Systemabsturz glaubt. Die implementierte Fortschrittsanzeige von drei Suchagenten ist in der Abbildung 8.2 abgebildet.

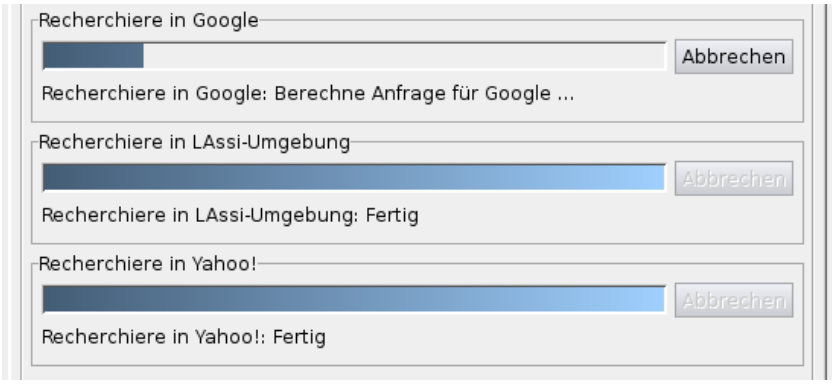

Abbildung 8.2: Fortschrittsanzeige der Suchagenten

Zur Implementierung dieser Fortschrittsanzeige ist das Beobachter-Entwurfsmuster nach Gamma et al. verwendet worden[1]. Im Beobachter-Muster sind zwei Rollen definiert: die des Beobachters und die des beobachteten Subjektes. Das Beobachter-Objekt implementiert eine Beobachter-Schnittstelle, welche eine Methode zur Benachrichtigung bei einer Zustandsänderung des Subjektes vorgibt. Es meldet sich über eine Methode am Subjekt an und übergibt sich selbst als Parameter. Das Subjekt speichert die Referenz auf den Beobachter ab und ruft die Benachrichtigungsmethode der Beobachter-Schnittstelle bei einer Zustandsänderung auf. Auf diese Weise wird der Beobachter über die Zustandsänderung des Subjektes informiert.

Die Abbildung 8.3 gibt in UML-Notation einen Überblick über die beteiligten Klassen. Objekte der Klassen “ProgressMonitorService” und “ProgressInformationMonitor” nehmen die Rolle des Subjektes und Objekte der Klassen “AgentProgressView” und “ProgressComposi-

[1] vgl. [4], Seite 287 ff.

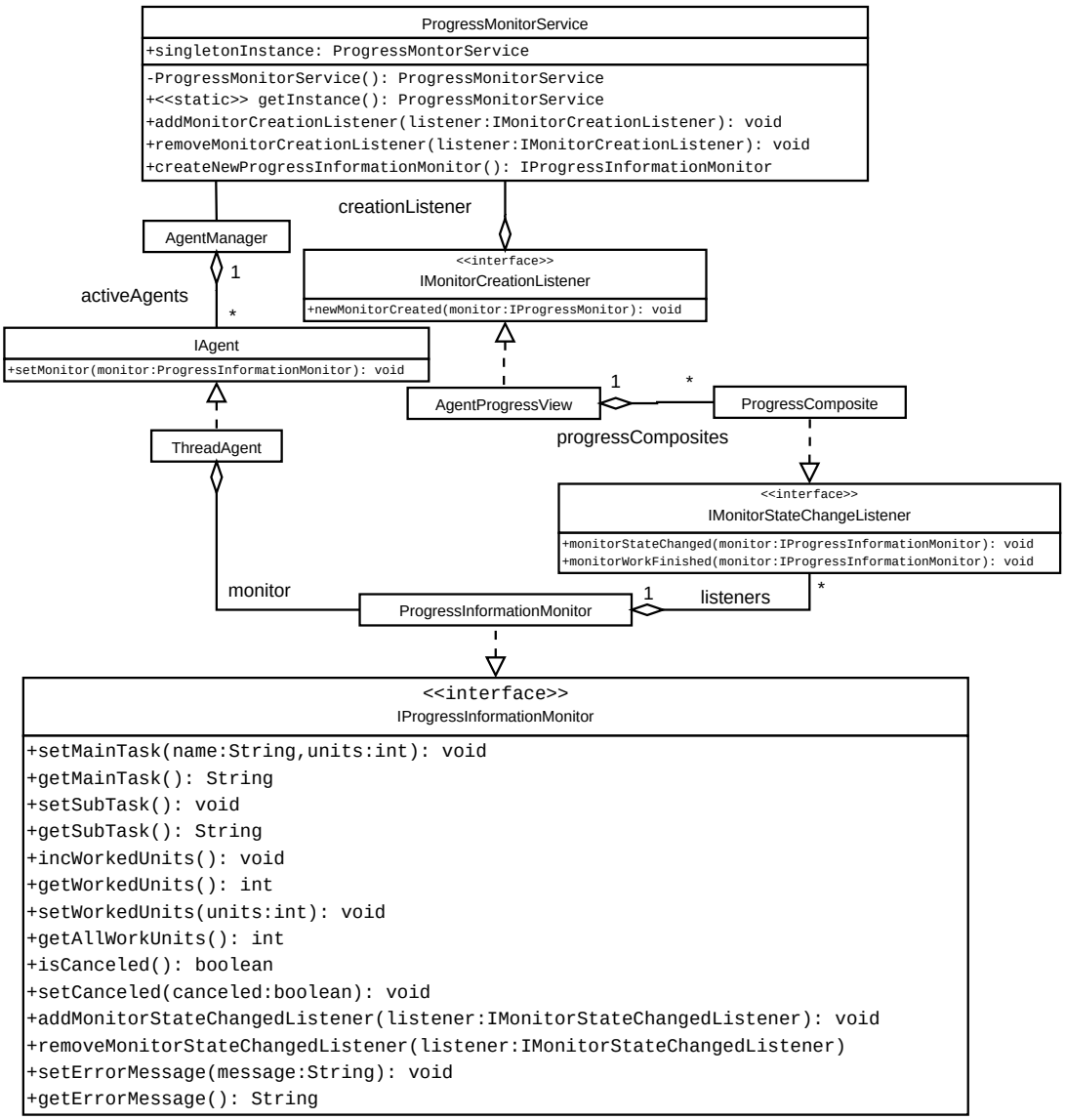

Abbildung 8.3: Fortschrittsanzeige der Suchagenten

te" die Rolle des Beobachters ein. Die Schnittstellen "IMonitorCreationListener" und "IMonitorStateChangeListener" sind die Beobachter-Schnittstellen. Die Singleton-Instanz der Klasse "AgentManager" erzeugt über ihre Agentenfabrik einen neuen Suchagenten. Die Suchagenten sind als Java-Threads, also als eigene Kindprozesse des LAssi-Prozesses im Betriebssystem, implementiert worden (Klasse "ThreadAgent"). Jedem Suchagenten wird über die Methode "setMonitor()" der Schnittstelle "IAgent" ein Objekt der Klasse "ProgressInformationMonitor" übergeben. Dieser Monitor dient als Abbild des Status des Suchagenten. Das heißt, dass der Suchagent Angaben bezüglich seines Zustands auf dem Monitor-Objekt setzt, z.B. die Anzahl der bereits recherchierten Treffer oder eine Hinweismeldung.

Ein Monitor wird von der Singleton-Instanz der Klasse "ProgressMonitorService" erzeugt. Sobald diese einen neuen Monitor erzeugt hat, informiert sie ihren Beobachter (Objekt der Klasse "AgentProgressView") über den neuen Monitor und übergibt ihn an den Beobachter. Die "AgentProgressView" ist das Fenster, welches den Agentenfortschritt anzeigt. Sie ordnet jedem Monitor ein Objekt der Klasse "ProgressComposite" zu, welches für die Darstellung des Monitorzustands in der "AgentProgressView" verantwortlich ist. Die Darstellung eines Monitors entspricht einem Kasten mit Fortschrittsbalken, Statuszeile und "Abbrechen"-Knopf, wie in der Abbildung 8.2. Das "ProgressComposite" soll als Beobachter des Monitors seinen Zustand optisch abbilden. Daher meldet es sich an dem übergebenen Monitor über die Methode "addMonitorStateChangedListener()" als Beobachter an. Ein beschriftetes Beispiel für ein "ProgressComposite"-Objekt zeigt die Abbildung 8.4.

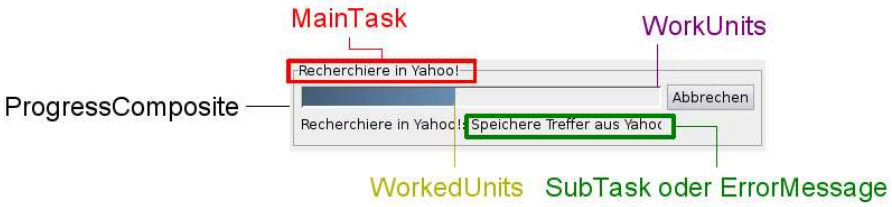

Abbildung 8.4: Komponenten eines "ProgressComposite"-Objektes

Vor dem Beginn seiner Recherche initialisiert der Suchagent sein Monitor-Objekt über die Methode "setMainTask()". Diese Methode erwartet als Parameter eine Bezeichnung des Monitors (Parameter "name") und die Anzahl der Arbeitsschritte, die vom Statusbalken angezeigt werden soll (Parameter "units"). Nach jedem Arbeitsschritt inkrementiert der Suchagent die Anzahl der bereits bearbeiteten Arbeitsschritte des Monitors über die Methode "incWorkedUnits()". Die Anzahl der Arbeitsschritte eines Suchagenten entspricht der Anzahl der zu recherchierenden Treffer plus eins. Die Anzahl der zu recherchierenden Treffer wird vom Schüler festgelegt. Der eine zusätzliche Schritt wird nur dazu benötigt, den Beginn der Arbeit

durch den Suchagenten anzuzeigen. Bei jeder Zustandsänderung des Monitor-Objektes ruft dieses die Methode “monitorStateChanged()” seiner Beobachter auf. Auf diese Weise wird das zum Monitor gehörende “ProgressComposite”-Objekt über die Zustandsänderung informiert, kann den neuen Monitor-Zustand (z.B. die Anzahl erledigter Arbeitsschritte) abrufen und sich neu zeichnen.

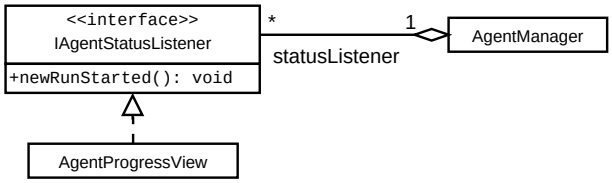

Abbildung 8.5: Beziehung zwischen “AgentManager” und “AgentProgressView”

Aber wie erhält die “AgentProgressView” die Information, dass ein neuer Recherchedurchgang begonnen hat und die “ProgressComposite”-Objekte der Suchagenten des vorherigen Durchgangs gelöscht werden können? Hier wird erneut das Beobachter-Muster angewendet, wie in Abbildung 8.5 dargestellt. Die Singleton-Instanz des “AgentManager” nimmt die Rolle des Subjektes ein und die “AgentProgressView” die Rolle des Beobachters. Zum Erzeugungszeitpunkt des Fensters “AgentProgressView” meldet sich dieses an der Singleton-Instanz über die Methode “addAgentStatusListener()” als Beobachter an. Sobald ein neuer Recherchedurchgang beginnt, also direkt vor der Erzeugung der neuen Suchagenten, wird die Methode “newRunStarted()” der “AgentStatusView” vom “AgentManager” aufgerufen. Die Implementierung der Methode in der Klasse “AgentStatusView” löscht alle “ProgressComposite”-Instanzen aus der Map, welche ein Monitor-Objekt einem “ProgressComposite”-Objekt zuordnet, und zeichnet sich anschließend neu. Auf diese Weise werden alle Darstellungen der “ProgressComposite”-Objekte aus dem Fenster entfernt.

8.3 Implementierung der Trefferansichten

Der Assistent bietet dem Schüler zwei Sichten auf die Inhalte einer Trefferschale: die Herkunfts- und die Clusteransicht. Jeder Ansicht ist je eine Registerkarte im Hauptfenster des Assistenten zugeordnet worden, sodass der Schüler komfortabel zwischen den Ansichten umschalten kann (siehe Abbildung 8.6).

Abbildung 8.6: Trefferansichten des Assistenten

8.3.1 Grafische Darstellung von Suchmaschinen und Treffern

Bei der Implementierung der Trefferansichten stellt sich die Frage, durch welche Objekte Suchmaschinen und Treffer grafisch repräsentiert werden. Aufgrund der Trennung von grafischer Benutzeroberfläche und Geschäftslogik enthalten die Klassen “Hit” bzw. “SearchService” und deren Subklassen keine Informationen zu Icons oder sonstigen optischen Darstellungsaspekten. Icons sind aber notwendig, weil der Schüler in den Trefferansichten schnell anhand eines Icons den Typ eines Treffers (z.B. PDF-Dokument, HTML-Seite oder LAssi-Karteikartendokument) oder seine Quellsuchmaschine erkennen können soll. Außerdem muss es über spezielle Objekte möglich sein, einen Treffer in einem geeigneten Viewer oder Editor zu öffnen. Neben der optischen ist auch die strukturelle Darstellung der Treffer entscheident. Treffer können eine Struktur beinhalten, z.B. ein LAssi-Desktop eine Menge von LAssi-Materialien. Die Anzeige dieser Materialien ist erforderlich, damit der Schüler sie öffnen und auswerten kann. Schließlich enthalten die LAssi-Materialien das vom Schüler aufbereitete Wissen und nicht der LAssi-Desktop.

Gelöst wird das Problem der grafischen Repräsentation durch die Implementierung von Adapterklassen für Objekte der Subklassen von “Hit” und “SearchService”. Zur Bereitstellung der Adapterklassen wird der Adaptermanager der Eclipse-Plattform verwendet. Von diesem Adaptermanager können Adapter für bestimmte Objekte abgerufen werden. Der Adaptermanager benötigt zur Instanziierung eines Adapters eine Instanz einer Fabrikklasse. Diese muss am Adaptermanager über den Erweiterungspunkt “org.eclipse.core.runtime.adapters” angemeldet werden. Neben Angabe der Fabrikklasse fordert der Adaptermanager auch die An-

gabe der adaptierten Klasse. Für die Fabrikklassen ist die Implementierung der Schnittstelle “IAdapterFactory” vorgegeben, welche die Methode “getAdapter()” zur Instanziierung eines Adapters definiert. Eine Adapterklasse muss von der abstrakten Klasse “org.eclipse.ui.model. WorkbenchAdapter” erben, welche z.B. die Implementierung von Methoden zur Bereitstellung eines Icons (“getImageDescriptor()”) und möglicher Substrukturen (“getChildren()”) fordert.

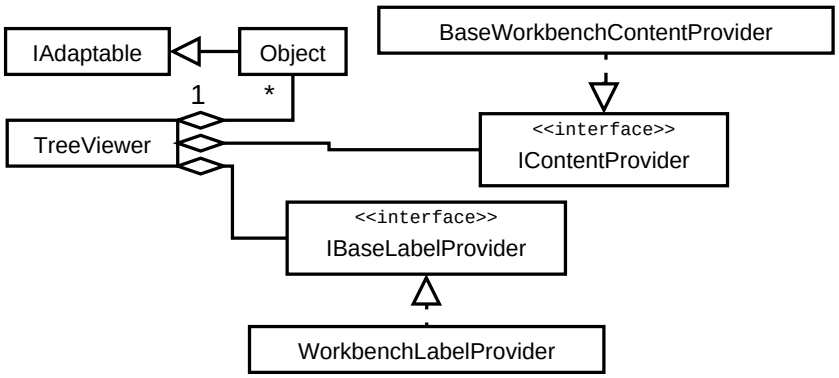

Abbildung 8.7: Trefferansichten mit dem JFace TreeViewer in UML-Notation

Da die Treffer sowohl in der Herkunfts- als auch in der Clusteransicht in einer Baumstruktur dargestellt werden können, wird zur Darstellung der Treffer das Steuerelement JFace TreeViewer verwendet. Wie in der Abbildung 8.7 dargestellt, benötigt ein “TreeViewer”-Objekt eine Struktur von Objekten, welche in der Baumansicht dargestellt werden sollen. Desweiteren wird ein Objekt einer Klasse benötigt, die die Schnittstelle “IContentProvider” implementiert (nicht zu verwechseln mit den Contentprovidern des Assistenten). Dieses Objekt ist für die Bereitstellung der Objektstruktur des Baumes zuständig. Anders als der “TreeViewer” kennt dieses Objekt die Typen der darzustellenden Objekte. Die grafische Darstellung der Knoten und Blätter des Baumes stellt ein Objekt einer Klasse bereit, welche die Schnittstelle “org.eclipse.jface.viewers.IBaseLabelProvider” implementiert. Dieses Objekt liefert z.B. die Darstellung der Knoten- bzw. Blattbeschriftung.

Das Eclipse-Framework RCP (Rich Client Platform) liefert für die Trefferansichten gut einsetzbare Standardimplementierungen für beide Schnittstellen: die Klassen “WorkbenchLabelProvider” und “BaseWorkbenchContentProvider”. Das Objekt der Klasse “BaseWorkbenchContentProvider” fragt direkt an den Objekten der anzuzeigenden Struktur ihr Adapterobjekt ab. Daher müssen diese die Schnittstelle “IAdaptable” implementieren, welche die Methode “getAdapter()” definiert. Hier liefern die Objekte dann ihren Adapter zurück, der über den Eclipse Adaptermanager erzeugt wird.

Der “WorkbenchLabelProvider” hat die Aufgabe, die Darstellung der Knotenbezeichnungen zu formatieren. In beiden Ansichten sollen die Titel der ungelesenen Treffer fett und die der gelesenen normal dargestellt werden. Alle anderen Knoten, z.B. Suchmaschinen- oder Clusterbezeichnungen, sind ebenfalls normal darzustellen. Daher wird die Klasse “WorkbenchLabelProvider” in der Implementierung der Trefferansichten anonym instanziiert und die Methode “getFont()” überschrieben. Diese Methode liefert ein Objekt der Klasse “Font”, welches Formatierungsangaben zur Bezeichnungsdarstellung, wie z.B. der Schriftart und deren Punktgröße enthält. Als Parameter erhält sie das Element des Baumes, für welches die Bezeichnungsdarstellung berechnet werden soll. Es muss anschließend mit dem “instanceof”-Operator geprüft werden, ob das Element vom Typ “Hit” ist. Falls dem so ist, dann ist dieses Element auf den Typ “Hit” zu casten. Anschließend kann über die Methode “isRead()” abgefragt werden, ob der Treffer bereits geöffnet worden ist. Ist er als gelesen markiert, so wird der Fontstyle auf fett, ansonsten auf normal gedruckt gesetzt. Die Bezeichnungen aller Elemente, die nicht vom Typ “Hit” sind, werden ebenfalls normal dargestellt.

8.3.2 Implementierung der Herkunftsansicht

Die Herkunftsansicht zeigt Treffer geordnet nach den Suchmaschinen an, von denen sie erzielt wurden. Durch die Herkunftsansicht kann der Schüler Treffer direkt Suchräumen zuordnen und deren Sichtung entsprechend priorisieren. Treffer aus der LAssi-Volltextsuche sind beispielsweise hochinteressant für den Schüler, weil er möglicherweise Wissen, welches er in einer anderen Aufgabenstellung selbst verstanden und aufbereitet hat, in der aktuellen Aufgabe erneut nutzen kann.

Die an der Herkunftsansicht beteiligten Klassen sind im UML-Diagramm in der Abbildung 8.8 dargestellt. Die Klasse “ServiceView” beinhaltet die Implementierung der Herkunftsansicht. Diese Klasse implementiert die Schnittstelle “IHitServiceChangeListener”. Daher kann sie von der “AssistantManagementView” als Beobachter für Zustandsveränderungen der Trefferschalen am “HitService” angemeldet werden.

Bei ihrer Initialisierung bzw. nach Zustandsänderungen im “HitService” ruft die Herkunftsansicht die aktuelle Trefferschale von der Singleton-Instanz des “HitService” über dessen Methode “getActiveBasin()” ab. Die Trefferschalen werden in der Implementierung durch Integer-Konstanten identifiziert. Mit dem erhaltenen Integer-Wert als Parameterwert wird über die Methode “getHitsOfBasin()” die Menge von Treffern der aktuellen Schale vom “HitService” abgerufen. Anschließend wird über die erhaltene Menge von “Hit”-Objekten iteriert und die Treffer über eine Map den Suchmaschinen zugeordnet (Schlüssel: “SearchService”, Wert: “List⟨Hit⟩”). Abschließend wird aus der Map eine Menge von “ServiceHitList”-Objekten erzeugt. Ein Objekt der Klasse “ServiceHitList” besitzt als Attribute eine Suchmaschine und

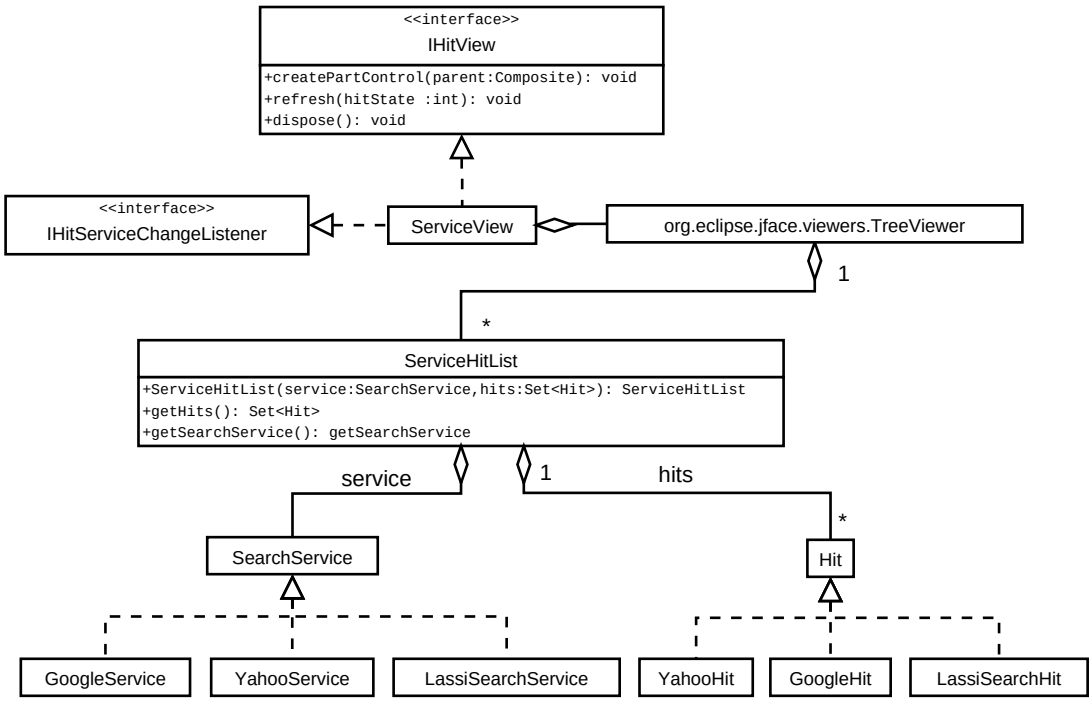

Abbildung 8.8: Beteiligte Klassen an der Berechnung und Darstellung der Herkunftsansicht

eine Menge von Treffern dieser Suchmaschine.

Die Herkunftsansicht verwendet zur grafischen Darstellung der Treffer ein “TreeViewer”-Objekt. Auf erster Ebene werden Suchmaschinen angezeigt. Um die Treffer einer Suchmaschine zu sehen, muss der Schüler den Zweig dieser Suchmaschine aufklappen. Er erhält anschließend die Ausgabe der Treffer dieser Suchmaschine untereinander. Die berechnete Menge von “ServiceHitList”-Instanzen wird dem “TreeViewer”-Objekt über dessen Methode “setInput()” übergeben. Als Adapter für die grafische Darstellung wird die abstrakte Klasse “org.eclipse.ui.model.WorkbenchAdapter” anonym mit drei überschriebenen Methoden instanziiert. Die Methode “getChildren()” liefert hier die Menge der “Hit”-Instanzen eines “Service HitList”-Objektes als Array, welches über die Methodenkomposition “getHits().toArray()” berechnet wird. “getLabel()” des “WorkbenchAdapter” liefert den Namen der Suchmaschine (Methode “getName()” der Klasse “SearchService”) und “getImageDescriptor()” ein Icon, welches der Suchmaschine zugeordnet werden kann.

8.3.3 Implementierung der Clusteransicht

Die Clusteransicht zeigt benannte Gruppen von inhaltlich ähnlichen Treffern an. Sie soll dem Schüler dabei helfen, die gesamte Treffermenge inhaltlich zu überblicken. Auf diese Weise kann der Schüler Zeit sparen, da er z.B. Cluster nach Sichtung weniger Treffer verwerfen kann, wenn diese Treffer geringwertig sind.

Das UML-Diagramm in der Abbildung 8.9 gibt einen Überblick über die beteiligten Klassen an der Clusteransicht. Die Clusteransicht selbst ist in der Klasse “SemanticClusterView” implementiert. Zur Berechnung der Cluster erzeugt die “SemanticClusterView” ein Objekt der Klasse “ClusterGenerator”. Der “ClusterGenerator” wird initialisiert mit einer Instanz der Klasse “KMeansClusteringFunction”, welche als Distanzfunktion die euklidische Distanz nutzt. Diese Distanzfunktion ist in einem Objekt der Klasse “EuklidDistanceFunction” gekapselt.

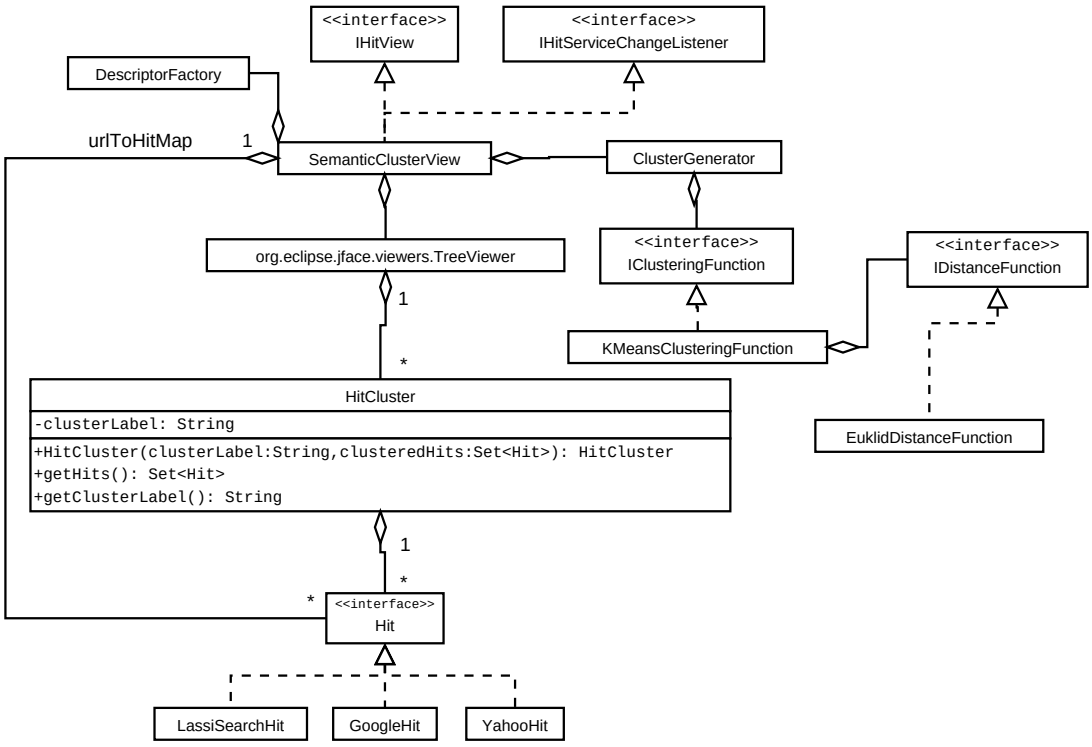

Abbildung 8.9: Beteiligte Klassen an der Berechnung und Darstellung der Clusteransicht

Die Methode “generateClusters()” der “ClusterGenerator”-Instanz liefert eine Menge von Mengen von URLs. Jeder URL muss anschließend auf das zugehörige “Hit”-Objekt abgebildet werden, die Cluster sollen schließlich Treffer enthalten. Hierzu hält die “SemanticClusterView” eine Map mit dem Bezeichner “urlToHitMap” (Schlüssel: String, Wert: “Hit”), welche

einem URL seinen Treffer zuordnet. Nach der 1:1-Abbildung der URL-Menge auf eine “Hit”-Menge, werden die Inhalte der “Hit”-Objekte über die Methode “getContents()” abgerufen und über die Methode “add()” an eine Instanz der Klasse “DescriptorFactory” gereicht. Diese berechnet nach Aufruf ihrer Methode “calculateDescriptors()” Deskriptoren für den Inhalt des aktuellen “Hit”-Clusters. Hierfür wird eine, für den Schüler konfigurierbare, Bewertungsfunktion für Token verwendet, beispielsweise die absolute, gewichtete Häufigkeit. Der erste berechnete Deskriptor wird als Label für das Cluster genutzt, wenn kein anderer Cluster so benannt worden ist. Ansonsten wird der nächste Deskriptor verwendet. Als Datenstruktur für Treffercluster werden Instanzen der Klasse “HitCluster” verwendet. Diese Klasse erwartet im Konstruktor die Parameter “clusterLabel” und ”clusteredHits”. Als Clusterlabel wird der von der “DescriptorFactory” berechnete Deskriptor verwendet. Als Wert des Parameters “clusteredHits” dient die aus der URL-Menge erzeugte “Hit”-Menge. Abschließend wird die Statistik der “DescriptorFactory” über deren Methode “clear” gelöscht, damit dieselbe Deskriptorenfabrik für die Berechnung des Labels des nächsten Clusters genutzt werden kann.

Die Clusteransicht verwendet ebenfalls eine Instanz der Klasse “TreeViewer” zur Darstellung der “HitCluster”-Objekte. Auf erster Ebene des Baumes werden die Cluster mit ihren Labels angezeigt. Sobald der Schüler ein Cluster öffnet, sollen ihm die “Hit”-Objekte des Clusters untereinander angezeigt werden. Als Adapter für die grafische Darstellung wird auch hier die abstrakte Klasse “org.eclipse.ui.model.WorkbenchAdapter” anonym mit drei überschriebenen Methoden instanziiert. Die Methode “getChildren()” liefert hier die Menge der “Hit”-Instanzen eines “HitCluster”-Objektes als Array, “getLabel()” das berechnete Clusterlabel und “getImageDescriptor()” ein spezifisches Icon. Die Abbildung 8.10 zeigt die Clusteransicht im praktischen Einsatz zum “Garnelen”-Beispiel.

Abbildung 8.10: Beispiel der Clusteransicht zum “Garnelen”-Beispiel

Bei der Implementierung tritt das Problem der Ladezeiten von Treffern beim Abruf ihrer Inhalte über die Methode “getContents()” auf. Sowohl das Laden von dem LAssi-USB-Stick des Schülers als auch aus dem WWW nimmt spürbar Zeit in Anspruch, auch bei kleinen Tref-

fermengen (ca. 10 bis 15 Treffer). Wenn das Laden der Trefferinhalte und die Berechnung der Cluster im LAssi-Betriebssystem-Prozess bearbeitet wird, dann reagiert die grafische Benutzeroberfläche während dieser Zeit nicht auf Benutzereingaben. Daher ist das Laden der Treffer und die Berechnung der Treffercluster in einen eigenen Thread ausgelagert worden. Dieser wird in der Klasse “SemanticClusterView” anonym instanziiert und berechnet die Cluster. Nach Abschluss der Clusterberechnung wird der LAssi-Prozess über den Abschluss informiert und stellt die berechneten Cluster dar. Um die benötigte Zeit für das Laden der Trefferinhalte zu senken, werden diese einmalig geladen und mit den Treffer-Metadaten ebenfalls persistiert. Offen ist in der aktuellen Implementierung, ob und wie die Trefferinhalte mit den tatsächlichen Inhalten der referenzierten Dokumente synchronisiert werden, z.B. mit Web-Seiten. Außerdem ist noch nicht definiert, für welchen Zeitraum die Trefferinhalte vorgehalten werden. Die Inhalte der Treffer kosten wervollen Speicherplatz auf dem LAssi-USB-Stick und bringen als Nutzen einen Geschwindigkeitsvorteil bei der Arbeit mit der Clusteransicht. Es muss hier geprüft werden, in welcher Intensität die Schüler tatsächlich im Unterricht mit der Clusteransicht arbeiten und ob der Speicherplatz den Geschwindigkeitsvorteil wert ist.

9 Systembewertung

Im Rahmen dieses Kapitels soll das entwickelte Konzept einer prozessorientierten Rechercheunterstützung für Schüler bewertet werden. Diese Bewertung soll auf Basis des implementierten Assistenten geschehen. Dazu werden zuerst Kennzahlen vorgestellt, anhand derer die Qualität der Trefferausbeute des Assistenten mit der von den Suchmaschinen Google und Yahoo verglichen werden kann. Um einen etwas umfassenderen Eindruck der Systemleistung zu gewinnen, ist eine Vorabversion des Assistenten an Schüler der Modellklasse in Hamburg-Haburg ausgeliefert worden. Die Rückmeldungen dieser Schüler werden im dritten Unterkapitel dargestellt. Anschließend erfolgt ein funktionaler Vergleich mit einer Suchmaschine am Beispiel von Google. Die Ergebnisse der Bewertungen werden abschließend in einm Fazit zusammengefasst und interpretiert.

9.1 Vorstellung von Qualitätsmaßen

Nach dem Benutzungsmodell definiert sich die Qualität der Leistung des Assistenten durch die Qualität der von ihm recherchierten Treffer. Das Benutzungsmodell sieht eine explizite Bewertung der recherchierten Treffer durch den Schüler vor. Auf diese Weise soll die Trefferqualität nachfolgender Suchdurchläufe der Suchagenten sukzessive verbessert werden. Es ist anzunehmen, dass die Trefferqualität eines Suchdurchlaufes durch die Bewertungen der Treffer als “hochwertig, zu berücksichtigen”, “hochwertig” und “geringwertig” repräsentiert wird. Insofern ist die Fehlerrate fr_i des Agenten für den i-ten Suchdurchgang durch den Anteil der “geringwertigen” Treffer dieses Suchdurchgangs gt_i an der Gesamtzahl der Treffer des Suchdurchgangs t_i definiert:

$$fr_i = \frac{gt_i}{t_i} \tag{9.1}$$

Der mittlere Fehler über n Suchdurchläufe ergibt sich aus dem geometrischen Mittel:

$$\overline{fr} = \sqrt[n]{\prod_{i=1}^{n} fr_i} = \sqrt[n]{\prod_{i=1}^{n} \frac{gt_i}{t_i}} \tag{9.2}$$

Neben der Fehlerrate soll auch der Anteil kontextbezogener Treffer ausgewertet werden. Dies misst die Retrieval-Kennzahl Precision, welche sich definiert durch:

$$prec_i = \frac{hbt_i + h_i}{t_i} \tag{9.3}$$

hbt_i liefere dabei die Anzahl der als "hochwertig, zu berücksichtigen" eingestuften und h_i die Anzahl der "hochwertigen" Treffer in der Ablageschale. Die mittlere Precision ergibt sich analog:

$$\overline{prec} = \sqrt[n]{\prod_{i=1}^{n} \frac{hbt_i + h_i}{t_i}} \tag{9.4}$$

Anzumerken ist, dass diese Kennzahlen nicht die Position eines Treffers in der Ergebnisliste berücksichtigen. Der Assistent oder z.B. Google werden nicht dafür belohnt, gute Treffer als erste Treffer anzuzeigen, sondern lediglich dafür die Treffer überhaupt angezeigt zu haben. Weiss et al. geben als weitere Kennzahlen zur Bewertung von IR-Systemen Recall und F-Measure an[1]. Deren Berechnung erfordert aber genaue Kenntnis der Indexinhalte der genutzten, externen Suchmaschinen. Da diese nicht vorliegt, können diese Kennzahlen nicht zur weiteren Bewertung des Assistenten herangezogen werden.

9.2 Bewertung anhand eines Fallbeispiels

Mit den vorgestellten Kennzahlen kann die Qualität der Rechercheergebnisse abgeschätzt werden. Daher soll anhand eines Fallbeispiels geprüft werden, ob der Assistent einem Schüler helfen kann, bessere Treffer als über die Web-Formulare von Google oder Yahoo zu recherchieren. Als Szenario für den Vergleich dient die bekannte Aufgabenstellung zum Thema "Garnelen" aus dem Biologieunterricht. Die Aufgabe des betrachteten Schülers lautet also, sich einen Überblick über die Thematik zu verschaffen, um anschließend ein Teilgebiet dieses Themas in einem Referat vor der Klasse vorzustellen.

Der Vergleich von Google und Yahoo mit dem Assistenten gestaltet sich schwierig. Der Grund hierfür liegt darin, dass Google und Yahoo anfrageorientiert mit dem Schüler interagieren: Der Schüler stellt eine Suchanfrage an das System und erhält eine Trefferausgabe. Als einzige weitere Interaktionmöglichkeit mit der Suchmaschine bleibt dem Schüler nur die Formulierung einer neuen Suchanfrage. Der Assistent unterstützt den Schüler hingegen im gesamten Suchprozess. Nach Ausgabe einer Liste von wenigen Treffern kann der Schüler durch die Bewertung der Treffer den Assistenten steuern und die Qualität der zukünftigen Tref-

[1] vgl. [12], Seite 79

fer somit erhöhen. Dies ist weder mit Google, noch mit Yahoo möglich. Es kann also nicht die Trefferausbeute einer einzigen Suchanfrage an Google oder Yahoo zur Berechnung der Kennzahlen herangezogen werden. Deshalb werden mehrere Anfragen an die Suchmaschinen gestellt und die berechneten Werte der Kennzahlen gemittelt.

Diese Art der Bewertung ist natürlich sehr subjektiv. Schließlich müssen Suchbegriffe erdacht und Treffer als hoch- oder geringwertig bewertet werden. Beide Faktoren sind entscheident vom Vorwissen des recherchierenden Schülers abhängig. Er verwendet die Suchbegriffe, die er mit der Aufgabenstellung assoziiert. Ein Treffer ist für ihn nur dann hochwertig, wenn er neue, für die Aufgabenstellung relevante, Informationen enthält. Um den Vergleich trotzdem realistisch zu gestalten, werden diejenigen Anfragen verwendet, mit denen die Schüler im Biologie-Unterricht tatsächlich ihre Recherchen begonnen haben. Da das Vorwissen eines Schülers nicht klar abgegrenzt werden kann, sei jeder Treffer hochwertig, der eine inhaltliche Beziehung zum Thema “Garnelen” hat, z.B. ein Steckbrief einer Süsswassergarnelenart. Geringwertig für den Schüler seien Einkaufsangebote, Rezepte, Forendiskussionen, sowie alle Treffer, die keine inhaltliche Beziehung zur Aufgabenstellung haben, z.B. das Profil eines Benutzers “Garnele” in einem Web-Forum.

Es existiert noch eine weitere Möglichkeit zum Vergleich des Assistenten mit Google und Yahoo. Betrachtet werden in dieser weiteren Variante nur die ersten zehn Treffer der Suchmaschinen. Auf Basis der Inhalte der Treffer werden neue Suchbegriffe gewählt und als neue Anfrage an die Suchmaschine gesendet. Dies entspricht im Prinzip der Suchstrategie des Assistenten. Es ist möglich, dass sich die Ergebnisse der Suchmaschinen und des Assistenten trotz derselben Strategie unterscheiden, weil die Statistik des Assistenten andere Begriffe auswählt als der Schüler. Trotzdem wird von dieser Variante abgesehen, weil sie nicht der bisherigen Suchstrategie der Schüler entspricht. Es soll das Verbesserungspotential der Recherchesituation der Schüler unter Einsatz des Assistenten abgeschätzt werden. Daher ist durch diesen Vergleich kein repräsentatives Ergebnis zu erwarten, denn die bisherige Suchstrategie der Schüler erhält in diesem Vergleich keinen Einzug.

Das Fallbeispiel soll folgendermaßen gestaltet werden: Der Schüler legt einen neuen LAssi-Desktop mit der Bezeichnung “Die Garnele” an. Er recherchiert mit dem Assistenten über vier Durchgänge zehn Treffer pro Durchgang. Nach jedem Durchgang werden geringwertige und hochwertige Treffer als solche vom Schüler bewertet. Die weitere Rechercherichtung wird vom Schüler über hochwertige, zu berücksichtigende Treffer gesteuert. Der Assistent soll pro Durchgang konstant mit einer Bewertungsfunktion für Token arbeiten (absolute Häufigkeit oder TF-IDF). Als Suchdienst verwendet der Assistent die Suchmaschine Yahoo, weil er für dieses System ein großes Verbesserungspotential erschließen kann.

Über die Web-Formulare von Google und Yahoo werden folgende Schüleranfragen gestellt (ohne Anführungszeichen):

- “Garnele” (Anfrage 1)
- “Garnelen” (Anfrage 2)
- “Garnele -rezepte -preis” (Anfrage 3)
- “Garnelen -rezepte -preis” (Anfrage 4)

Zu den Anfragen 3 und 4 sei gesagt, dass die Schüler über die Filterbegriffe versucht haben, die ungewünschten Rezepte und Einkaufsangebote zu filtern. Außerdem ist zu berücksichtigen, dass die Schüler erst nach einer Einweisung in den logischen NICHT-Operator in der Lage waren, diese Suchanfragen zu formulieren. Da in einem Durchgang des Assistenten der Inhalt von drei Treffern nicht abgerufen werden konnte (aufgrund der Nicht-Erreichbarkeit der Webserver), werden nur die ersten 37 Treffer des Assistenten und der Suchmaschinen betrachtet. Außerdem werden redundant angezeigte Treffer der Suchmaschinen wie neue Treffer behandelt. Wenn zwei verschiedene Suchanfragen (z.B. Anfrage 3 und 1) denselben hochwertigen Treffer liefern, dann wird dieser für beide Suchanfragen als hochwertig gezählt.

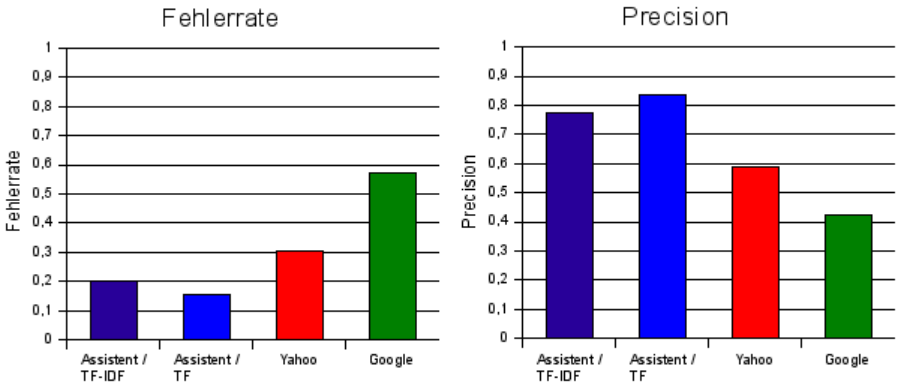

Abbildung 9.1: Gemittelte Messwerte für Fehlerrate und Precision

Die gemittelten Messwerte für die Fehlerrate und die Precision sind in der Abbildung 9.1 gegenübergestellt worden. Es zeigt sich, dass die Fehlerraten des Assistenten mit 20% (TF-IDF) und 15% (absolute Häufigkeit) geringer sind, als die von Google mit 57% und Yahoo mit 31%. Hier muss ergänzt werden, dass beide Suchmaschinen ihre hochwertigen Treffer als erstes angezeigt haben. Da diese Suchmaschinen über keine Rückmeldemöglichkeit verfügen, werden ab dem ca. 20. Treffer viele völlig irrelevante Treffer angezeigt. Besonders Google tat

sich hervor, denn Google lieferte hier auffallend viele Treffer mit kommerziellem Hintergrund oder von interaktiven Plattformen für z.B. den Austausch von Videos (Clipfish, MyVideo, YouToube, ...). Da die genaue Funktionsweise von Google nicht dokumentiert ist, kann an dieser Stelle über die Gründe für dieses Resultat nur spekuliert werden. Im Prinzip nutzt Google als Bewertungskriterium eines Treffers hauptsächlich die Anzahl anderer Webseiten, die diesen Treffer über einen Link referenzieren. Oft verlinkte Webseiten erscheinen daher weit oben auf der Trefferliste. Dahinter steht die Annahme, dass der Inhalt häufig referenzierter Dokumente wahrscheinlich für viele Benutzer interessant ist. Gerade die Anbieter kommerzieller Inhalte schalten häufig Werbelinks zum eigenen Angebot auf anderen Webseiten (z.B. mit einem Werbebanner). Deshalb besitzen viele andere Seiten Referenzen auf das kommerzielle Angebot und beeinflussen dessen Bewertung durch Google.

Es stellt sich die Frage, welche Fehlerrate bzw. Precision Google und Yahoo bei präziseren Suchbegriffen statt der Filterbegriffe in Anfrage 3 und 4 liefern. Daher sollen die Werte des Assistenten denen von Google und Yahoo bei präziseren Suchbegriffen in einem zweiten Versuch gegenübergestellt werden. Anfrage 3 und 4 werden daher im zweiten Versuch durch die folgenden Anfragen ersetzt:

- “Nordseegarnelen” (Anfrage 5)
- “Zwerggarnelen” (Anfrage 6)

Die Messung wird für die Suchmaschinen mit den neuen Anfragen (insg. also Anfragen 1, 2, 5, 6) wiederholt. Es werden wieder die ersten 37 Treffer betrachtet. Die Ergebnisse sind in der Abbildung 9.2 dargestellt. Das Ergebnis der ersten Messreihe hat sich als relativ stabil erwiesen. Yahoo konnte mit den präziseren Anfragen seinen Precision-Wert um 8% auf 67% erhöhen, Google hingegen nur von 42% auf 44%. Die Fehlerraten haben sich entsprechend gemindert. So erreicht Yahoo eine Fehlerrate von 23% (vorher 30%) und Google von 55% (vorher 57%). Auch hier muss ergänzt werden, dass die Qualität der Treffer von Google erst auf den letzten zwei Ergebnisseiten stark gesunken ist.

Die Messreihen zeigen exemplarisch, dass der implementierte Assistent den Schülern mehr hochwertige Treffer liefern konnte, als Google oder Yahoo zu ihren Anfragen erzielten. Natürlich gibt es Szenarien, in denen der Assistent schlechter abschneidet als Google oder Yahoo. Ungeachtet dessen konnte das Potential des Assistenten demonstriert werden. Abschließend ist zu sagen, dass Google oder Yahoo genau dann eine große Menge hochwertiger Treffer liefern können, wenn die Schüler präzise Suchbegriffe in das Web-Formular der Suchmaschinen eingeben, z.B. den lateinischen Fachbegriff der Nordseegarnele “crangon crangon”. Zur Assoziation derlei präziser Suchbegriffe ist aber ein umfangreiches Vorwissen notwendig, über

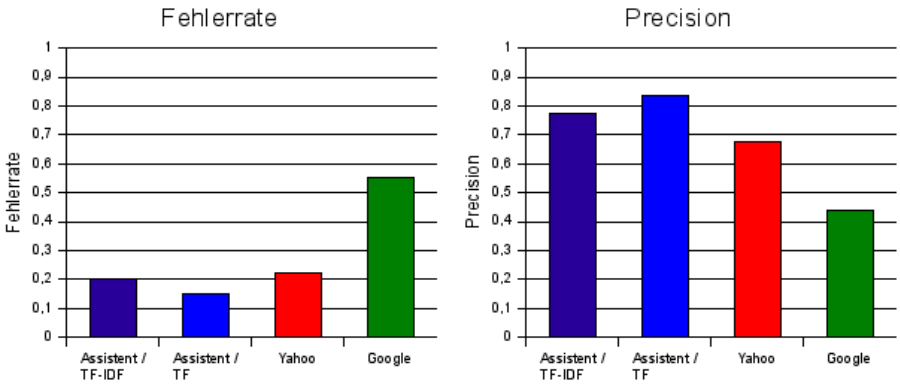

Abbildung 9.2: Gemittelte Messwerte für Fehlerrate und Precision

welches die Schüler zu Beginn ihrer Einarbeitung in ein neues Themengebiet schlicht nicht verfügen. Daher ist der Assistent auch nicht als Alternative, sondern als Ergänzung zur Nutzung von Google und Yahoo zu sehen. Der Assistent hilft den Schülern, sich eine Informationsgrundlage zu beschaffen. Bei der Auswertung und Verarbeitung der Assistententreffer ergeben sich meistens konkrete Fragestellungen, wie z.B. “Wie entwickelt sich denn eigentlich der Bestand der Nordseegarnele crangon crangon in der Nordsee über die letzten Jahre?” Aus diesen Fragestellungen können die Schüler selbst Suchbegriffe ableiten, mit denen sie über Google eine präzise Trefferausbeute erhalten, z.B. mit der Anfrage “bestand crangon crangon nordsee”.

Abschließend soll noch einmal darauf hingewiesen werden, dass in diesem Szenario nur die Rechercheleistung des Assistenten gemessen worden ist. Sein Mehrwert durch die integrierte LAssi-Volltextsuche (Erinnerungsfunktion) und die zwei Trefferansichten (Herkunfts- und Clusteransicht), sowie durch die automatische Archivierung der erzielten Treffer sind nicht betrachtet worden.

9.3 Bewertung anhand eines Schuleinsatzes

Fünf Schüler der befragten achten Klasse haben sich bereit erklärt, während einer vollständigen Unterrichtseinheit den Assistenten auszuprobieren. Dazu ist ihnen eine Vorabversion bereitgestellt worden, welche noch nicht über die Clusteransicht und persistente Trefferschalen verfügt. Als Suchmaschinen wurden die LAssi-Volltextsuche und die Suchmaschine Yahoo integriert. Zum Abschluss der Unterrichtseinheit nach ca. fünf Schulwochen sind die Schüler mit einem Fragebogen und in freien Interviews zu ihrem Eindruck von dem Assistenten befragt worden.

Von allen befragten Schülern hat der Assistent für seine Leistung die Schulnote 2 erhalten. Sie bestätigten die hohe Anzahl hochwertiger Treffer, die ihnen “ihr” Assistent geliefert hat. Negativ sind ihnen die zeitweise langen Recherchezeiten und die in der Vorabversion fehlende Schalenpersistenz aufgefallen. Die Schüler bestätigten die Fähigkeit des Assistenten, sich “trainieren” zu lassen. Konkret bedeutet das, dass die Schüler die Rechercherichtung des Assistenten durch Trefferbewertungen gut steuern konnten und für sie gut brauchbare Treffer erzielt worden sind.

Besonders hervorgehoben haben die Schüler die Zeitersparnis durch die Suche des Assistenten im Hintergrund. Während sie an ihrer Aufgabenstellung gearbeitet haben, recherchierte der Assistent automatisch Treffer und präsentierte diese nach Abschluss seiner Recherchen. In der Zeit, in der die Schüler die Treffer des Assistenten ausgewertet und deren Informationen auf LAssi-Materialien übertragen haben, recherchierte der Assistent weitere hochwertige Treffer für die Schüler. Auf diese Weise mussten sie ihren Arbeitsprozess nicht unterbrechen. Die Filterleistung des Assistenten bewerteten die Schüler im Durchschnitt mit 7.8 Punkten auf einer Skala von 1 bis 10 Punkten (10 Punkte seien die bestmögliche Bewertung). Die Steuerbarkeit des Assistenten in verschiedene inhaltliche Richtungen erhielt durchschnittlich 7.0 Punkte.

Besonders negativ ist den Schülern auch aufgefallen, dass der Assistent für LAssi-Desktops mit vielen LAssi-Materialien Treffer zu bereits abgeschlossenen Teilaufgaben liefert. Die LAssi-Materialien zu abgeschlossenen Teilaufgaben enthalten meistens eine besonders umfangreiche Menge an Texten. Daher beeinflussen diese Materialien die statistischen Verfahren zur Berechnung von Suchbegriffen in der aktuellen Implementierung stark.

Interessant ist außerdem gewesen, dass einige Schüler sich spezielle Recherche-Desktops angelegt haben. Auf diesen Desktops befand sich eine einzige LAssi-Karteikarte, die Suchbegriffe in der Form enthalten hat, wie sie auch direkt in die Web-Formulare der Suchmaschinen eingegeben werden können. Auf diese Weise konnten die Schüler das Bewertungssystem für Treffer ohne die Berücksichtigung der LAssi-Materialien auf dem LAssi-Desktop ihrer aktuellen Aufgabenstellung nutzen. Die Vorstellung, den Assistenten “trainieren” zu können, hat die Schüler in hohem Maße zu seiner Nutzung motiviert. Daher nutzten sie den Assistenten auch für Recherchen, die sie über die Web-Formulare der Suchmaschinen hätten tätigen können.

9.4 Funktionaler Vergleich mit einer Suchmaschine am Beispiel Google

Dieses Unterkapitel vergleicht den implementierten Assistenten direkt mit der Suchmaschine Google. Der Assistent nutzt weiterhin die Suchmaschine Yahoo. Der Vergleich basiert ebenfalls auf dem “Garnelen”-Beispiel.

Die Schüler der Modellklasse begannen ihre Recherchen mehrheitlich mit der Anfrage “Garnelen” bei Google. Auf diese Anfrage lieferte Google ihnen unter den ersten zehn nur vier nützliche Treffer: einen Wikipedia-Artikel mit dem Titel “Garnele” und die Homepages mehrerer Aquaristikliebhaber mit Haltungstipps für Süsswassergarnelen und Krebse. Die restlichen Treffer umfassten Links zu Aquaristikforen, zu Internetshops für Zierfische und zu Rezeptsammlungen. Auf der zweiten Ergebnisseite von Google fanden sich drei Links zu Homepages von Garnelenzüchtern, welche ebenfalls Haltungstipps beinhalteten und zwei Links zu einer Artendatenbank mit Steckbriefen zu verschiedenen Garnelenarten. Zusammenfassend lässt sich sagen, dass unter den ersten 20 Treffern neun grundsätzlich für die Schüler nützlich gewesen sind. Lediglich zwei Treffer (Wikipedia-Artikel) enthielten Informationen z.B. zur einheimischen Nordseegarnele, mit denen die Schüler weitere Recherchen abseits der Aquaristik starten könnten.

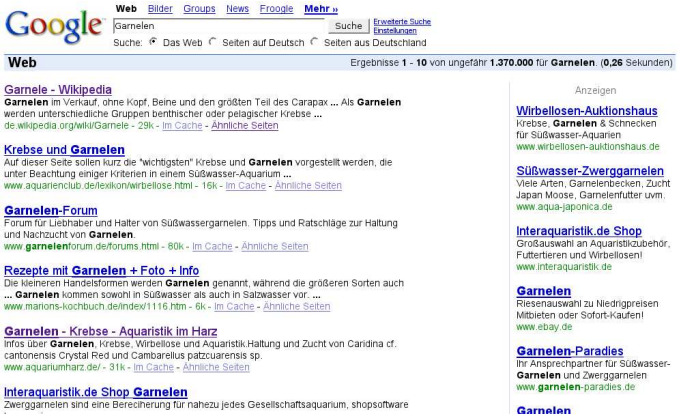
Google
Web Bilder Groups News Froogle Mehr »
Garnelen
Suche
Erweiterte Suche
Einstellungen
Suche: Das Web Seiten auf Deutsch Seiten aus Deutschland
Web
Ergebnisse 1 - 10 von ungefähr 1.370.000 für Garnelen. (0,26 Sekunden)
Garnele - Wikipedia
Garnelen im Verkauf, ohne Kopf, Beine und den größten Teil des Carapax ... Als Garnelen werden unterschiedliche Gruppen benthischer oder pelagischer Krebse ...
de.wikipedia.org/wiki/Garnele - 29k - Im Cache - Ähnliche Seiten
Krebse und Garnelen
Auf dieser Seite sollen kurz die "wichtigsten" Krebse und Garnelen vorgestellt werden, die unter Beachtung einiger Kriterien in einem Süßwasser-Aquarium ...
www.aquarienclub.de/lexikon/wirbellose.html - 16k - Im Cache - Ähnliche Seiten
Garnelen-Forum
Forum für Liebhaber und Halter von Süßwassergarnelen. Tipps und Ratschläge zur Haltung und Nachzucht von Garnelen.
www.garnelenforum.de/forums.html - 80k - Im Cache - Ähnliche Seiten
Rezepte mit Garnelen + Foto + Info
Die kleineren Handelsformen werden Garnelen genannt, während die größeren Sorten auch ... Garnelen kommen sowohl in Süßwasser als auch in Salzwasser vor. ...
www.marions-kochbuch.de/index/1116.htm - 6k - Im Cache - Ähnliche Seiten
Garnelen - Krebse - Aquaristik im Harz
Infos über Garnelen, Krebse, Wirbellose und Aquaristik.Haltung und Zucht von Caridina cf. cantonensis Crystal Red und Cambarellus patzcuarensis sp.
www.aquariumharz.de/ - 31k - Im Cache - Ähnliche Seiten
Interaquaristik.de Shop Garnelen
Zwerggarnelen sind eine Bereicherung für nahezu jedes Gesellschaftsaquarium, shopsoftware
Anzeigen
Wirbellosen-Auktionshaus
Krebse, Garnelen & Schnecken für Süßwasser-Aquarien
www.wirbellosen-auktionshaus.de
Süßwasser-Zwerggarnelen
Viele Arten, Garnelenbecken, Zucht Japan Moose, Garnelenfutter uvm.
www.aqua-japonica.de
Interaquaristik.de Shop
Großauswahl an Aquaristikzubehör, Futtertieren und Wirbellosen!
www.interaquaristik.de
Garnelen
Riesenauswahl zu Niedrigpreisen Mitbieten oder Sofort-Kaufen!
www.ebay.de
Garnelen-Paradies
Ihr Ansprechpartner für Süßwasser-Garnelen und Zwerggarnelen
www.garnelen-paradies.de
Garnelen

Abbildung 9.3: Treffer von Google zur Anfrage “Garnelen”

Welche Möglichkeiten haben die Schüler nun, um ihre Trefferausbeute zu verbessern? Google bietet den Schülern über die grafische Benutzeroberfläche keine Möglichkeit, die Rezeptsammlungen und Aquaristikforen bzw. die Haltungstipps zu filtern. Es existiert zwar ein Link “Ähnliche Seiten”, der direkt unter jedem Treffer angezeigt wird, aber auch dieser hilft den Schülern wenig. Zu dem Wikipedia-Artikel, also dem besten gelieferten Treffer, klassifiziert Google elf andere Seiten als ähnlich. Unter diesen befinden sich fünf Wörterbuchseiten,

welche Synonyme oder Reime zu einem Begriff finden. Ansonsten wird ein Link zu einem Wikipedia-Spendenaufruf und einer zur Publikationslizenz “GNU Free Documentation License” angezeigt. Unter den restlichen vieren befindet sich lediglich ein einziger neuer Treffer, der vorher noch nicht angezeigt worden ist und sich mit den ökologischen und sozialen Folgen der Garnelenzucht u.a. in Indien befasst. Der letzte Treffer könnte als einziger von den ähnlichen Seiten für Schüler interessant sein, da er sich direkt auf die industrielle Nutzung der Garnelen bezieht.

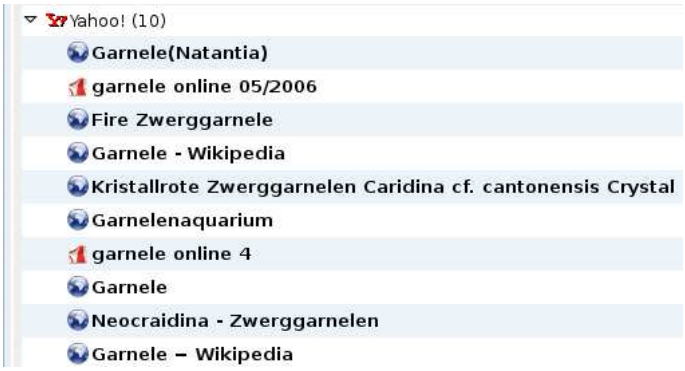

Abbildung 9.4: Rechercheergebnisse im ersten Durchgang des Assistenten

Als weitere Möglichkeit zur Verbesserung der Trefferausbeute bleibt den Schülern noch die Nutzung des logischen NICHT-Operators, welchen aber keiner der Schüler bei den Befragungen kannte. Angenommen, dieser Operator wäre bekannt gewesen, so stünden die Schüler immernoch vor dem Problem, Filterbegriffe zu wählen. Die Einkaufsangebote könnten mit “-shop”, die Aquaristikforen mit “-aquarium” und die Rezeptsammlungen mit “-rezept” gefiltert werden. Die resultierende Anfrage “garnelen -shop -aquarium -rezept” liefert aber immernoch vier Links zu Rezeptsammlungen, welche erst durch den Zusatz “-rezepte” gefiltert werden. Die vier nun freigewordenen Plätze werden von Google aufgefüllt mit Ebay-Angaboten und Links zu Preisvergleichen. Letztendlich bleibt den Schülern an dieser Stelle nur die Möglichkeit erst die Wikipedia-Artikel als einzige nützliche Treffer auszuwerten und auf Basis dieser Auswertung eine neue Anfrage an Google zu richten. Diese könnte zum Beispiel die lateinischen Fachbegriffe bestimmter Garnelenarten erhalten, welche zu einer sehr präzisen Trefferausbeute führen. Allerdings ist die Formulierung derlei Anfragen den Schülern erst möglich, wenn sie bereits über eine Informationsgrundlage verfügen. Interessant ist außerdem, dass die Anfrage “Garnele” statt “Garnelen” bei Google zu einer wesentlich besseren Trefferausbeute geführt hätte, aber keiner der Schüler diese Anfrage formuliert hat. Die Bezeichnungen der LAssi-Desktops der Schüler enthielten den Begriff “Garnele” hingegen fast überall. Die Schüler verwenden bei der Formulierung von Fließtexten viele Begriffe in bestimmten For-

men (z.B. Fällen bei Nomen), die eine hohe Trefferausbeute bei Google erzielen würden. Diese Begriffe geben sie in dieser Form aber selten in das Web-Formular von Google ein.

Nachdem die Schüler beispielsweise den Desktop “Die Garnele” angelegt haben, beginnt der Assistent sofort mit seinen Recherchen. Das Resultat der ersten Recherche enthält deutlich mehr nützliche Treffer zu Garnelen als die erste Ergebnisseite von Google, weil hier Treffer zu konkreten Garnelenarten präsentiert werden (siehe Abbildung 9.4). Die Schüler haben nun die Möglichkeit, durch einfache Trefferbewertungen über ein Kontextmenü mehr Informationen zu einem Themengebiet zu erhalten. Angenommen, der Schüler möchte weitere Informationen zur Gattung “Neocaridina” der Zwerggarnelen haben, klickt er einfach mit der rechten Maustaste auf den Treffer “Neocaridina - Zwerggarnelen” und verschiebt diesen Treffer in die Schale der “hochwertigen, zu berücksichtigenden” Treffer. Das Ergebnis des folgenden Recherchedurchgangs gibt die Abbildung 9.5 wieder.

Abbildung 9.5: Rechercheergebnisse im zweiten Durchgang des Assistenten

Unter diesen Treffern befindet sich lediglich ein unpassender Treffer, auf denen Verkaufsangebote für Süsswassergarnelen zu finden sind. Dieser Treffer wird, ebenfalls über das Kontextmenü, in die Schale der unerwünschten Treffer verschoben. Auf diese Weise soll das Erscheinen von derlei Treffern zukünftig unterbunden werden. Unter den Treffern des dritten Recherchedurchganges befinden sich erneut sieben für den Schüler sehr nützliche Links zur Zweggarnele.

Der große Mehrwert des Assistenten besteht darin, dass der Schüler ihn intiutiv über die Bewertung von Treffern in verschiedene Rechercherichtungen steuern kann, ohne dabei eine einzige Anfrage zu formulieren. Desweiteren archiviert der Assistent die Treffer über die Schalen automatisch für den Schüler. Dieser behält, aufgrund der optischen Hervorhebung von ungelesenen Treffern, auch über umfangreiche Trefferschalen die Übersicht.

9.5 Fazit

Sowohl das Fallbeispiel, als auch die Befragung der Schüler zeigen, dass der gewählte Ansatz mit guten Leistungen die Recherchesituation der Schüler optimiert. Die Schüler finden durch die Unterstützung mehr hochwertige Treffer als über die Web-Formulare von Google und Yahoo. Außerdem arbeiten sie aufgrund der "Trainierbarkeit" ihres Assistenten hochmotiviert mit der Software.

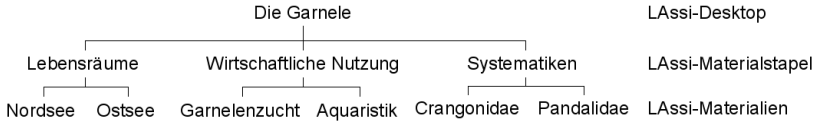

Abbildung 9.6: Beispiel für den Inhalt eines LAssi-Desktops

Der gewählte Ansatz birgt aber noch viel Potential. Die Schüler berichteten, dass ihr Assistent ihnen bei umfangreichen LAssi-Desktops viele Treffer zu bereits bearbeiteten Teilaufgaben ihrer Aufgabenstellung lieferte. Wie ist das zu erklären? Die Analyse eines beispielhaften Schülerdesktops hat ergeben, dass sich die vom Schüler erstellten LAssi-Materialien auf einen eng abgegrenzten Themenbereich der Aufgabenstellung beziehen. Ein Beispiel für den Inhalt eines Schülerdesktops zeigt die Abbildung 9.6. Zum Verständnis sei gesagt, dass LAssi-Materialstapel spezielle LAssi-Materialien sind, welche eine Liste von anderen LAssi-Materialien beinhalten. Die Schüler sammeln ihre LAssi-Materialien zu einem Themengebiet bzw. einer Teilaufgabe ihrer Aufgabenstellung in LAssi-Materialstapeln. Zu Teilaufgaben, die die Schüler bereits bearbeitet haben, existieren umfangreichere LAssi-Materialien als zu offenen Teilaufgaben. Daher erhalten Token von den abgeschlossenen Teilaufgaben stärker Einzug in die häufigkeitsbasierte Statistik zur Auswahl von Suchbegriffen, als Token von nicht abgeschlossenen Teilaufgaben. Sie sind schlicht öfter auf den LAssi-Materialien notiert worden.

Aus dieser Beobachtung erschließt sich ein weiteres Problem: der Assistent gewinnt seine Suchbegriffe aus dem gesamten Inhalt des aktuellen LAssi-Desktops und den hochwertigen, zu berücksichtigenden Treffern. Daher kann der Fall eintreten, dass der Assistent mit Suchbegriffen aus verschiedenen Themengebieten gleichzeitig sucht, z.B. mit der Anfrage "garnele UND lebensräume UND nordsee UND ostsee UND "wirtschaftliche nutzung" UND garnelenzucht UND aquaristik UND systematiken UND crangonidae UND pandalidae". Diese Anfragen sind meist so streng, dass sie keine oder nur sehr wenig Treffer liefern. Die Treffer der abgeschwächten Anfragen passen meist nicht zum aktuell bearbeiteten Themengebiet, z.B. der Systematik Crangonidae. Die Suchbegriffe können also nur bei wenig umfangreichen LAssi-Desktops aus dem gesamten Desktopinhalt sinnvoll gewonnen werden.

Zur Lösung beider Probleme müsste der Assistent die Themengebiete bzw. Teilaufgaben

des aktuellen LAssi-Desktops kennen. Außerdem müsste er wissen, welche LAssi-Materialien zu welcher Teilaufgabe gehören und an welcher Teilaufgabe der Schüler gerade arbeitet. Auf dem LAssi-Desktop könnten Themengebiete mit dem vorgestellten und implementierten Clustering-Verfahren k-Means berechnet werden. Jeder Cluster würde dabei als Themengebiet interpretiert. Als aktuell vom Schüler bearbeitetes Themengebiet könnte derjenige Cluster herangezogen werden, der die meisten, zuletzt bearbeiteten, LAssi-Materialien beinhaltet. Die Qualität der Suchbegriffe könnte noch erhöht werden, wenn auch die hochwertigen, zu berücksichtigenden Treffer in die Clusterberechnung einbezogen würden. Überschreitet die Anzahl der Materialien eines Clusters einen Schwellwert, so muss dieses Cluster durch Berechnung von Subclustern in seine Themengebiete unterteilt werden. Die Suchbegriffe sind dann aus dem Subcluster mit den meisten zuletzt bearbeiteten LAssi-Materialien zu extrahieren.

Auf diese Weise kann der Assistent Taxonomien (IST-EIN-Beziehungen) in Form von IST-TEILGEBIET-VON-Beziehungen innerhalb der Materialien und Treffern der Schüler erkennen, ohne dass der Schüler diese semantisch annotieren müsste. Eine solche Rechercheunterstützung ist vor allem vor dem Hintergrund der Entwicklungen um das “soziale Web” oder “Web 2.0” interessant. Das beschriebene Konzept bietet den Ansätzen des “Semantic Web” gegenüber den Vorteil, auf völlig unstrukturierten und nicht annotierten Datenmengen eine taxonische Struktur erkennen zu können. Suchassistenten, die auf dem “Semantic Web” basieren, sind auf die vorherige manuelle Modellierung von Beziehungen zwischen den Materialien des Suchraumes angewiesen.

10 Abschlussbetrachtungen

10.1 Anregungen für weitere Entwicklungen

Während der Entwicklung des beschriebenen Assistentensystems sind weitere Ideen aufgekommen, die im Rahmen dieser Arbeit nicht berücksichtigt werden konnten. Diese Ideen sind in diesem Kapitel gesammelt worden und sollen Anregungen für weitere Entwicklungen des beschriebenen Assistenten geben.

- **Automatische Erkennung von Themengebieten auf dem LAssi-Desktop:** Ab einem gewissen Umfang der LAssi-Materialien auf einem LAssi-Desktop beeinflussen diese Materialien die statistischen Auswahlverfahren für Suchbegriffe signifikant, sodass dem Schüler genau zu den Themengebieten weitere Treffer angezeigt werden, deren Bearbeitung er bereits abgeschlossen hat. Der beschriebene Lösungsansatz für dieses Problem besteht in der Berechnung von inhaltlichen Clustern der Materialien auf dem Desktop und der Beschränkung der statistischen Suchbegriffsauswahl auf jeweils den Cluster, in dem die meisten zuletzt bearbeiteten LAssi-Materialien liegen.

- **Lokaler Trefferredundanzfilter:** Der entwickelte Assistent identifiziert einen Treffer anhand seines URL und schließt auf Basis dieser Idenfifikation die doppelte Anzeige von Treffern aus. Häufig werden im Internet dieselben Dokumente auf verschiedenen Servern gespeichert, z.B. Artikel der Online-Enzyklopädie Wikipedia. Daher werden den Schülern trotz URL-Filter gleiche Treffer mehrfach angezeigt. Hier wäre die Entwicklung eines lokalen Redundanzfilters denkbar, der neue Treffer ab einem gewissen statistischen Ähnlichkeitswert trotz neuer URL blockt.

- **Minimierung des Deskriptorredundanz in Suchanfragen:** Wie bereits beschrieben, wird bei der Auswahl von Such- und Filterbegriffen im aktuellen System nicht berücksichtigt, dass Deskriptoren in anderen Deskriptoren enthalten und damit in bestimmten Konstellationen irrelevant für die Trefferausbeute sein können. Es wäre interessant ein Verfahren zu entwickeln, welches diese Deskriptorredundanz in Anfragen minimiert oder ausschließt.

- **Automatisches Streben nach minimaler Fehlerrate:** Im aktuellen Systementwurf muss der Schüler selbst eines der vorgestellten statistischen Bewertungsverfahren für Token (absolute, gewichtete Häufigkeit oder TF-IDF) auswählen. Die Schüler haben die Anpassungsmöglichkeit im praktischen Systemeinsatz gar nicht bis sehr wenig genutzt. Es wäre hier sinnvoll, wenn der Agent die im vorherigen Kapitel definierte Fehlerrate misst und nach Minimierung des gemessenen Wertes strebt. Hierfür könnte der Assistent z.B. lernen, mit welchem Verfahren er in den meisten Situationen eine minimale Fehlerrate erzielt.

- **Kontextglobale Identifizierung geringwertiger Treffer:** Die freien Interviews haben ergeben, dass die Schüler ihren Assistenten pro Aufgabenstellung über Trefferbewertungen immer neu mitteilen müssen, dass sie sich für Treffer mit bestimmten Inhalten nicht interessieren, z.B. für Einkaufsangebote von Onlineshops. Es wäre hier hilfreich, wenn der Assistent lernen könnte, was einen geringwertigen Treffer unabhängig der konkreten Aufgabenstellung ausmacht. Auf diese Weise würden die Schüler noch einmal Zeit sparen, weil sie ihren Assistenten nicht für jede Aufgabenstellung neu trainieren müssen.

- **Experten- statt Dokumentsuche:** In der entwickelten Form sucht der Assistent nach Trefferdokumenten in verschiedenen Suchräumen. Die Konsequenz für den Schüler hieraus ist die, dass er sich mit den gefundenen Treffern den Lernstoff selbst aneignen muss. Im Sinne eines kollaborativen Lernszenarios, in dem die Schüler sich gegenseitig Wissen vermitteln, wäre eine Expertensuche äußerst sinnvoll. Expertensuche heißt, dass der Assistent dem Schüler sagen kann, welche Mitschüler sich schon einmal mit einer Aufgabenstellung beschäftigt haben, die einen inhaltlichen Bezug zur aktuellen Aufgabenstellung des Schülers hat. Auf diese Weise weiß der Schüler, welche Mitschülerin oder welchen Mitschüler er bei Verständnisfragen ansprechen kann. Realisiert werden könnte eine solche Funktion in einem Computernetzwerk mit mehreren aktiven LAssi-Lernumgebungen, welche über einen Freigabemechanismus für LAssi-Materialien verfügen. Außerdem muss jede LAssi-Instanz den Namen ihres jeweiligen Schülers kennen. Der Assistent benötigt abschließend eine Suchmaschine, welche die freigebenen LAssi-Materialien in einem solchen LAssi-Netzwerk durchsuchen kann. Eine Skizzierung des Ergebnisses einer solchen Suche zum “Garnelen”-Beispiel ist in der Abbildung 10.1 dargestellt. Der Assistent weist den Schüler darauf hin, dass seine Mitschüler Anna und Sven sich ebenfalls in anderen Zusammenhängen mit “Garnelen” beschäftigt haben.

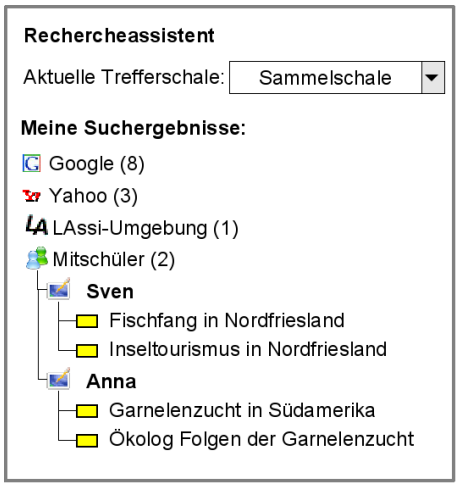

Abbildung 10.1: Expertensuche im Klassenverband

10.2 Zusammenfassung, Fazit und Ausblick

Die Aufgabe dieser Diplomarbeit bestand in der Konzeption einer Rechercheunterstützung für Schüler bei der Suche nach digital verfügbaren Informationen, welche in die Lernumgebung LAssi integriert werden sollte. Zur Lösung dieser Problemstellung ist nach Analyse der aktuellen Recherchesituation von Schülern ein Prozessmodell eines software-gestützten Rechercheprozesses für Schüler entwickelt worden. Außerdem wurde ein Anfragenkatalog definiert, welcher Vorgaben an das zu konzipierende Software-System stellt. Auf Basis dieser Anforderungen und des entwickelten Prozessmodells wurde eine Assistenten-Software für Recherchen entworfen und in das LAssi-System implementiert. Die Leistung dieses Assistenten wurde anhand messbarer Kennzahlen und anhand eines praktischen Schuleinsatzes bewertet. Abschließend sind kurz Räume für weitere Arbeiten skizziert worden.

Mit dem Ansatz eines dezentralen, lokalen Assistenten kann die Recherchesituation von Schülern stark optimiert werden. Es ist den Schülern mit dem entwickelten System möglich ihren Suchprozess, ohne Kenntnis aussagenlogischer Operatoren und konkreter Anfragensyntax einer Suchmaschine, zu steuern. Die fehlende Steuerungsfunktion bei Suchmaschinen, z.B. in Form von über die grafische Oberfläche bedienbaren Filtern, stellt für viele Schüler eine schwer überwindbare Hürde dar. Die mit dem Assistenten erzielten Treffer werden zusätzlich automatisch persistiert, womit er eine Archivierungsfunktion übernimmt. Dennoch birgt dieser Ansatz noch viel Potential für weitere Entwicklungen. Besonders die automatische Erkennung von Themengebieten innerhalb einer Aufgabenstellung birgt ein großes Optimierungspotential der Trefferqualität des Assistenten, da so ein noch feinerer thematischer Bezug der Treffer des Assistenten zur aktuellen Teilaufgabenstellung besteht.

A Benutzerhandbuch

Um die Aufgaben aus deinem Schulunterricht zu lösen, musst du sicher häufig im World Wide Web recherchieren, oder? Dann hast du bestimmt auch schon die Erfahrung gemacht, dass es ganz schön schwierig sein kann, die richtigen Suchbegriffe für die Suchmaschinen im Web zu finden, richtig? Und zu guter Letzt nimmt die Auswertung der langen Trefferlisten auch viel deiner Zeit in Anspruch, stimmts? Dein LAssi-System kann dich ab sofort bei deinen Recherchen unterstützen. Es beinhaltet einen Assistenten, der für dich automatisch zu deiner aktuellen Aufgabenstellung recherchiert. Dieses Handbuch stellt dir deinen Rechercheassistenten vor und zeigt dir, wie du mit ihm arbeiten kannst.

A.1 Ablaufbedingungen

Der entwickelte Rechercheassistent benötigt LAssi-Desktop in der Version 2.1. Wie die LAssi-Umgebung auch, ist der Assistent unter den Betriebssystemen Windows 98, ME, 2000 und XP, MacOS 10.3 und Linux lauffähig. Für die Linux-Installation ist zu beachten, dass die Grafikbibliotheken GTK+ 2.2.1 oder alternativ Open Motif 2.1 installiert sein müssen. Zusätzlich wird die Installation einer Java-Laufzeitumgebung in der Version 1.5 vorausgesetzt. Nachfolgerversionen der genannten Komponenten können bei 100%iger Kompatibilität ebenfalls eingesetzt werden.

A.2 Programminstallation und -start

Der Assistent muss als ein fester Bestandteil des LAssi-Systems nicht gesondert installiert werden. Nach Start dem des LAssi-Systems ist der Assistent ebenfalls automatisch geladen worden.

A.3 Bedienungsanleitung

Dein Rechercheassistent hilft dir bei der Recherche nach Informationen zu deiner aktuellen Aufgabe aus dem Schulunterricht. Wie macht er das? Ganz einfach: Er sucht für dich im World Wide Web und in deinen LAssi-Desktops vollautomatisch nach Treffern. Woher bekommt der Assistent die Suchbegriffe, mit denen er recherchiert? Er geht davon aus, dass du für jede Aufgabe einen eigenen Desktop in deinem LAssi-System anlegst. Aus den LAssi-Materialien auf diesem Desktop wählt er diejenigen Wörter aus, die besonders oft vorkommen. Schau dir den Assistenten doch einmal an.

A.3.1 Öffnen des Assistentenfensters

Du kannst das Hauptfenster des Assistenten über die in der Abbildung A.1 dargestellte Schaltfläche öffnen. Dazu klickst du mit der linken Maustaste einmal auf das markierte Symbol.

Abbildung A.1: Schaltfläche zum Öffnen des Hauptfensters

Das Hauptfenster ist in der Abbildung A.2 dargestellt. Alles, was du mit dem Assistenten machen kannst, ist über dieses Fenster möglich. Über die Schaltfläche mit dem grünen Pfeilsymbol kannst du deinen Assistenten ein- und ausschalten. Du kennst das Symbol sicher von deinem CD- oder MP3-Player. Ob dein Assistent ein- oder ausgeschaltet ist, siehst du in der rot markierten Statuszeile. Sobald du deinen Assistenten eingeschaltet hast, beginnt er mit seiner ersten Recherche.

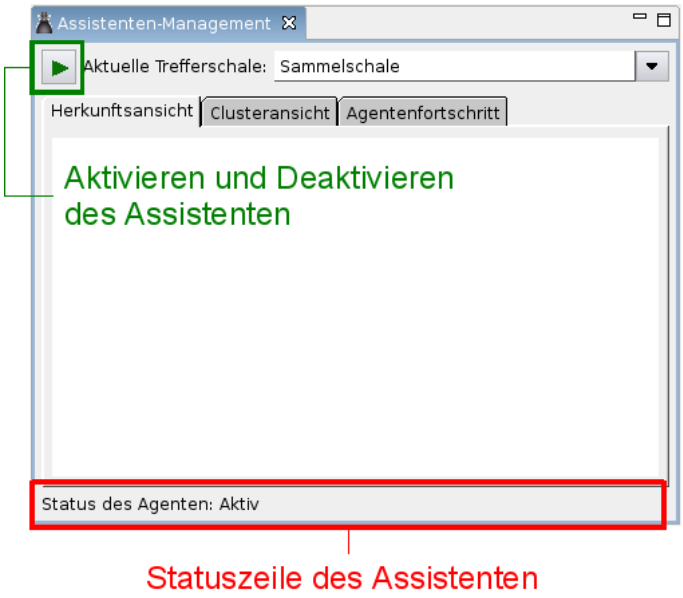

Abbildung A.2: Hauptfenster des Assistenten

Sicher interessiert dich jetzt, wie du den Assistenten steuern kannst? Woher soll er auch wissen, in welche inhaltliche Richtung deiner Aufgabenstellung er zu recherchieren hat?

A.3.2 Recherchen mit dem Assistenten

Der Assistent stellt dir für jede Aufgabenstellung, also für jeden LAssi-Desktop, genau vier Schalen bereit. In diese Schalen kannst du die Treffer des Assistenten einsortieren. Der Assistent wählt dann aus bestimmten Steuerschalen genau wie vom aktuellen LAssi-Desktop Begriffe aus, mit denen er sucht. Das ist schon alles. Und welche Schalen gibt es genau?

- Schale für hochwertige Treffer

  In diese kommen die Treffer, zu deren Inhalt du gern mehr erfahren möchtest. Du merkst schon, dies ist eine "Steuerschale". Eine "Steuerschale" ist eine Schale, aus der der Assistent zusätzlich zum Desktop Suchbegriffe heraussucht. Wichtig ist, dass du die Treffer wieder aus dieser Schale nimmst, wenn der Assistent in die inhaltliche Richtung dieser Treffer nicht mehr recherchieren soll.

- Ablageschale

  Diese Schale ist deine persönliche Trefferablage. Hier kannst du alle Treffer ablegen, die du für die Bearbeitung deiner Aufgabenstellung gebrauchen kannst. Der Assistent schaut sich die Treffer in dieser Schale gar nicht an.

- Sammelschale

  In diese Schale legt der Assistent seine recherchierten Treffer ab. Von hieraus musst du sie dann in eine der anderen drei Schalen verschieben.

- Abfallschale

  In diese Schale kannst du alle Treffer verschieben, die dich gar nicht interessieren oder die gar nicht zu deiner Aufgabenstellung passen. Diese Schale ist auch eine "Steuerschale", denn der Assistent wählt aus dieser Schale ebenfalls Begriffe aus. Dies sind aber keine Such-, sondern Filterbegriffe. Er recherchiert nach Treffern, die alle Suchbegriffe und keinen Filterbegriff enthalten.

Soweit, so gut. Die Treffer kommen also in Schalen. Aber wie kann man sich die Schalenінhalte und vor allem die Treffer ansehen?

A.3.3 Ansicht von Trefferschalen

Stelle dir einmal vor, dass du dich im Biologie-Unterricht mit dem Thema "Garnelen" auseinander setzen musst. Du sollst beispielsweise einen Vortrag zu einer Garnelenart deiner Wahl vor der Klasse halten. Zu dieser Aufgabenstellung legst du einen LAssi-Desktop mit der Bezeichnung "Die Garnele" an. Dein Assistent beginnt sofort mit seiner ersten Recherche. Sobald er seine Recherchen abgeschlossen hat, informiert er dich über eine kleine Sprechblase über der angezeigten Uhr deines Betriebssystems (siehe Abbildung A.3).

Im Hauptfenster deines Assistenten ist standardmäßig die Sammelschale geöffnet. Oben rechts im Hauptfenster siehst du ein Menü, welches sich nach nach einem Klick mit der linken Maustaste auf den Pfeil aufklappt. Du erhältst hier eine Übersicht über alle Trefferschalen, aus der du diejenige auswählen kannst, die du ansehen möchtest (siehe Abbildung A.4).

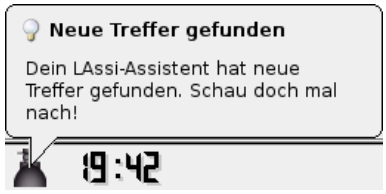

Abbildung A.3: Benachrichtigung des Assistenten über abgeschlossene Recherche

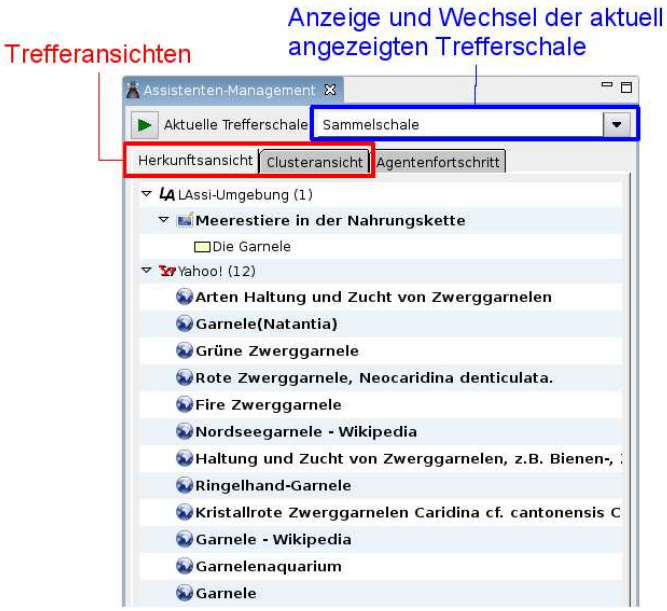

Abbildung A.4: Hauptfenster des Assistenten nach abgeschlossener Recherche

Der Assistent stellt dir zwei verschiedene Ansichten für die Schalen bereit: die Herkunftsansicht und die Clusteransicht. Du kannst die Ansicht für die aktuelle Trefferschale über die zwei Registerkarten mit den Beschriftungen “Herkunftsansicht” und “Clusteransicht” auswählen. Die Herkunftsansicht zeigt dir eine Übersicht der befragten Suchmaschinen, über die der Assistent Treffer recherchiert hat. Der jeweiligen Suchmaschine sind ihre Treffer untergeordnet. So kannst du schnell erkennen, woher ein Treffer stammt. Du solltest z.B. Treffer der LAssi-Suchmaschine immer als erstes anschauen. Vielleicht kannst du ja deine Lösung aus einer anderen Aufgabenstellung in die aktuelle Aufgabe einbringen. In der Abbildung A.4 sind Treffer in der LAssi-Suchmaschine und in Yahoo gefunden worden. Standardmäßig ist die Herkunftsansicht ausgewählt.

Die Abbildung A.5 zeigt die zweite Ansicht einer Trefferschale: die Clusteransicht. Ein Cluster ist eine Gruppe von Treffern. Alle Treffer in einem Cluster sind sich inhaltlich sehr ähnlich. Deshalb hat der Assistent die Treffer in ein gemeinsames Cluster einsortiert. Zusätzlich hat der Assistent versucht, einen Namen für jedes Cluster zu finden, der dessen Inhalt beschreibt. Die Clusteransicht hilft dir, einen Überblick über die gesamte Treffermenge zu

Abbildung A.5: Clusteransicht des Assistenten nach abgeschlossener Recherche

erhalten, ohne dass du jeden Treffer öffnen musst. Du kannst anhand der Bezeichnungen entscheiden, welches Cluster für dich am interessantesten ist. Wenn du dir ein paar Treffer eines Cluster angeschaut hast und diese thematisch unbrauchbar sind, dann ist es sehr wahrscheinlich, dass die restlichen Treffer dieses Clusters ebenfalls unbrauchbar sind. Du kannst dieses dann verwerfen. Du wirst dich jetzt sicher fragen, wie du z.B. einen Treffer öffnest, oder?

A.3.4 Arbeit mit Treffern und Steuern des Assistenten

Egal in welcher Ansicht du dich befindest, die Treffer werden immer gleich gehandhabt. Du kannst einen Treffer öffnen, indem du ihn doppelt mit der linken Maustaste anklickst. Du wirst beobachten, dass sich die Darstellung dieses Treffers in der jeweiligen Trefferansicht ändert. Vor dem Öffnen wird die Schrift des Treffertitels fett und nach dem Öffnen normal angezeigt. Vielleicht kennst du ein ähnliches Verhalten von deinem E-Mail-Programm: Es gibt dort gelesene und ungelesene Mails. Genauso verhält es sich auch mit Treffern: es gibt gelesene und ungelesene Treffer. Auf diese Weise siehst du schnell, welche Treffer du noch sichten musst.

Wenn du mit der rechten Maustaste auf einen Treffer klickst, dann erscheint ein Kontextmenü. Dieses Menü beinhaltet weitere Operationen, die du auf dem Treffer ausführen kannst, z.B. das Markieren eines Treffers als gelesen bzw. ungelesen. Außerdem kannst du den Treffer auch aus der aktuellen Trefferschale löschen. Der Assistent fragt dich anschließend, ob du den Treffer wirklich löschen möchtest. Wenn du diese Sicherheitsabfrage bejahst, wird der Treffer unwiderbringlich aus der Trefferschale entfernt. Bitte beachte, dass der Assistent dir diesen Treffer für den aktuellen LAssi-Desktop nie wieder anzeigen wird. Zu anderen Desktops wird der Treffer aber wieder gefunden - es sei denn, du hast ihn auch dort gelöscht.

Du kannst den Assistenten in seinen zukünftigen Recherchen steuern, indem du Treffer in die Schale für hochwertige Treffer bzw. in die Abfallschale verschiebst. Möchtest du beispielsweise mehr zur Nordseegarnele wissen, so verschiebst du den gefundenen Wikipedia-Artikel in die Schale der hochwertigen Treffer. Sollte dich die Zucht von Zwerggarnelen überhaupt

nicht interessieren, so wäre dieser Treffer in die Abfallschale zu befördern. Nachdem dein Assistent dir ausreichend viele Treffer zur Nordseegarnele geliefert hat, solltest du diesen Treffer aus der Schale der hochwertigen Treffer in die Ablageschale verschieben. Stattdessen verschiebst du nun diejenigen Treffer in die hochwertige Schale, die sich auf das nächste Themengebiet deiner Aufgabenstellung beziehen, dass dich interessiert (z.B. zur grünen Zwerggarnele).

A.3.5 Anzeige des Recherchefortschritts

Du kannst deinen Assistenten bei seinen Recherchen beobachten. Dazu holst du einfach die Registerkarte “Agentenfortschritt” mit einem Mausklick in den Vordergrund (siehe Abbildung A.7). Hier siehst du für jeden Arbeitsschritt des Assistenten mitsamt seinem aktuellen Status. Du kannst die einzelnen Arbeitsschritte mit einem Mausklick auf die Schaltfläche “Abbrechen” beenden.

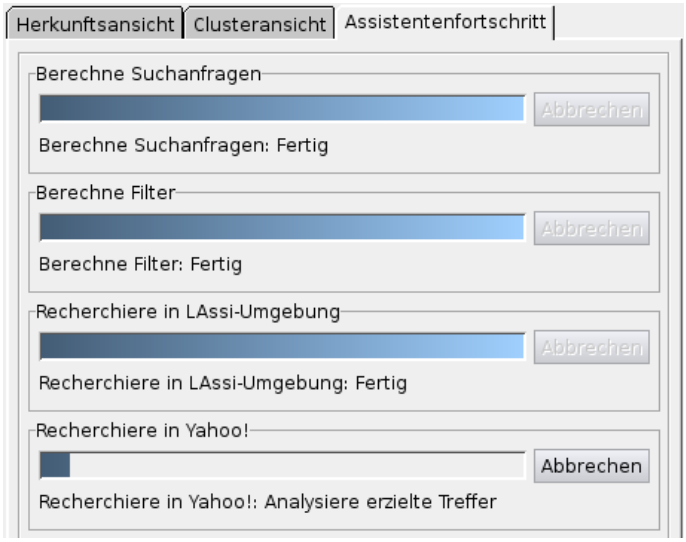

Abbildung A.6: Fortschrittanzeigen des Assistenten

A.3.6 Konfiguration des Assistenten

Der Assistent bietet dir verschiedene Konfigurationsmöglichkeiten. Du öffnest den Konfigurationsdialog über den Eintrag “Benutzervorgaben” des Menüs “Fenster”. Die Optionen des Assistenten befinden sich dort in der Optionsgruppe “Assistent”. Nachdem du die gewünschten Einstellungen vorgenommen hast, musst du auf die Schaltfläche “OK” unten links im Optionsdialog klicken. Die Einstellungen werden dann gespeichert und das Fenster geschlossen. Wenn du das Fenster verlassen möchtest, ohne dass die geänderten Einstellungen gespeichert werden, musst du auf die Schaltfläche “Abbrechen” klicken.

Optionsgruppe “Assistent”

Du kannst hier direkt angeben, wieviele Treffer der Assistent pro Suchdurchgang pro Suchmaschine recherchieren soll. Außerdem ist es möglich anzugeben, wieviele Treffer sich in der Sammelschale befinden sollen. Wenn die hier angegebene Anzahl von Treffern erreicht ist, dann ersetzt der Assistent ungelesene Treffer aus der Sammelschale willkürlich durch neue. Erst wenn sich nur gelesene Treffer in der Sammelschale befinden, bricht der Assistent seine Recherchen ab. Er wird dir erst dann wieder Treffer liefern, wenn du Platz in der Sammelschale geschaffen hast. Der Assistent wartet zwischen seinen Recherchen eine bestimmte Zeit. Du kannst in diesem Optionsdialog einstellen, wieviele Minuten das genau sein sollen.

Optionsgruppe “Filter”

In der Optionsgruppe “Filter” kannst du angeben, wieviele Filterbegriffe aus der Abfallschale berechnet werden sollen. Außerdem ist es hier möglich anzugeben, wie ähnlich sich zwei Wörter sein müssen, um als gleich zu gelten. Beispiel: Für die Begriffe “Kinder” und “Kindes” wird eine Ähnlichkeit von 80% berechnet.Wenn du den entsprechenden Schieberegler auf 80% einstellst, dann nutzt der Assistent einen der Begriffe als Suchbegriff, ansonsten beide.

Optionsgruppe “Suchanfragen”

In dieser Optionsgruppe sind alle Einstellungsmöglichkeiten zusammengefasst, die deinen Assistenten bei der Auswahl von Suchbegriffen beeinflussen.

- Im Kasten **“Finden von Suchbegriffen”** kannst auswählen, mit welchem mathematischen Verfahren dein Assistent die Suchbegriffe auswählt. Das obere Verfahren nimmt einfach die Begriffe, die in allen deinen LAssi-Materialien und den Treffern der hochwertigen Schale vorkommen. Das untere Verfahren wählt hingegen die Begriffe aus, die in einem Material häufig und in den restlichen sehr selten vorkommen.

- Im Kasten **“Identifizierung von ähnlichen Suchbegriffen”** kannst angeben, ob der Assistent versuchen soll, den gemeinsamen Stamm zweier Begriffe zu erkennen. “Der Kinder” und “des Kindes” wird dann beispielsweise als “Kind” erkannt. Du musst beachten, dass dein Assistent in diesem Fall den Begriff “Kind” auch als Suchbegriff nutzt.

- Im Kasten **“Anzahl der Suchbegriffe in einer Anfrage”** kannst du einstellen, mit wievielen Suchbegriffen dein Assistent maximal suchen soll. Hierzu klickst du einfach den Schieberegler mit der linken Maustaste an, hältst die Taste gedrückt und ziehst den Regler nach links bzw. rechts. Die aktuell eingestellte Begriffsanzahl wird dir direkt unter dem Regler angezeigt.

- Im Kasten **“Finden von Suchbegriff-Kombinationen”** kannst du auswählen, wieviele Wörter dein Assistent maximal miteinander kombinieren soll. Der Assistent setzt diese Kombinationen in Anführungszeichen und gibt sie bei den Suchmaschinen ein. Das bewirkt, dass alle Treffer genau diese Wortfolge enthalten müssen.

Optionsgruppe “Clusteransicht”

In dieser Optionsgruppe kannst du eingeben, wieviele Cluster dein Assistent in der Clusteransicht bilden soll.

A.4 Fehlermeldungen

Bei ungewöhnlich langen Recherchen, solltest du einmal einen Blick auf den Inhalt der Registerkarte “Assistentenfortschritt” werfen. Sämtliche Fehlermeldungen, die in Zusammenhang mit den externen Suchmaschinen stehen, werden hier angezeigt (siehe Abbildung).

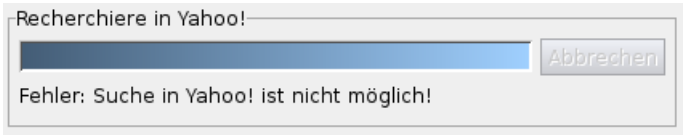

Abbildung A.7: Fehlermeldung des Assistenten

In diesem Fall überprüfe bitte die Netzwerkverbindung zur Suchmaschine, z.B. über das Internet. Sollte die Verbindung stabil sein, so überprüfe die Erreichbarkeit der Suchmaschine, in dem du einfach manuell eine Anfrage an die Suchmaschine über deinen Browser sendest.

A.5 Wiederanlaufbedingungen

Sollte das Programm, z.B. durch einen Stromausfall, unterbrochen werden, so existieren keine besonderen Wiederanlaufbedingungen.

B Protokolle der statistischen Untersuchungen

Die folgenden Suchanfragen sind am 13.02.2007 im Zeitraum von 16:00Uhr bis 18:00Uhr abgesendet worden.

Produkt	Durchgang i / Anfrage	t_i	gt_i	$hbt_i + h_i$	$prec_i$	fr_i
Assistent / TF-IDF	1	10	1	9	0,9	0,1
	2	10	2	8	0,8	0,2
	3	8	2	6	0,75	0,25
	4	9	3	6	0,67	0,33
Summe / Mittel	-	**37**	**8**	**29**	**0,78**	**0,2**

Produkt	Durchgang i / Anfrage	t_i	gt_i	$hbt_i + h_i$	$prec_i$	fr_i
Assistent / TF	1	10	1	9	0,9	0,1
	2	9	2	7	0,78	0,21
	3	9	1	8	0,89	0,11
	4	9	2	7	0,78	0,22
Summe / Mittel	-	**37**	**6**	**31**	**0,84**	**0,15**

Produkt	Durchgang i / Anfrage	t_i	gt_i	$hbt_i + h_i$	$prec_i$	fr_i
Yahoo	1	37	7	30	0,81	0,19
	2	37	23	14	0,38	0,62
	3	37	5	32	0,86	0,14
	4	37	20	17	0,46	0,54
Mittel	-	**37**	**13,75**	**23,25**	**0,59**	**0,31**

Produkt	Durchgang i / Anfrage	t_i	gt_i	$hbt_i + h_i$	$prec_i$	fr_i
Yahoo	1	37	7	30	0,81	0,19
	2	37	23	14	0,38	0,62
	5	37	3	34	0,92	0,08
	6	37	10	27	0,73	0,27
Mittel	-	**37**	**10,75**	**26,25**	**0,67**	**0,23**

Produkt	Durchgang i / Anfrage	t_i	gt_i	$hbt_i + h_i$	$prec_i$	fr_i
Google	1	37	20	17	0,46	0,54
	2	37	23	14	0,38	0,62
	3	37	19	18	0,49	0,51
	4	37	23	14	0,38	0,62
Mittel	-	**37**	**21,25**	**15,75**	**0,42**	**0,57**

Produkt	Durchgang i / Anfrage	t_i	gt_i	$hbt_i + h_i$	$prec_i$	fr_i
Google	1	37	20	17	0,46	0,54
	2	37	23	14	0,38	0,62
	5	37	17	20	0,54	0,46
	6	37	22	15	0,41	0,59
Mittel	-	**37**	**20,5**	**16,5**	**0,44**	**0,55**

C Integration einer Suchmaschine am Beispiel Yahoo

An dieser Stelle soll kurz die Integration einer externen Suchmaschine in den Assistenten am Beispiel von Yahoo beschrieben werden. Yahoo stellt zur Integration seiner Suchfunktion eine Java-Bibliothek im Internet bereit[1]. Auf die Wiedergabe der Adapterklasse für die grafische Repräsentation der Treffer und der Suchmaschine wird verzichtet. Da die Adapterklasse über einen Eclipse Erweiterungspunkt in das System eingebunden wird, wird dieses Konzept zu Beginn kurz skizziert.

C.1 Eclipse Erweiterungspunkte

Im Eclipse-Kontext werden ganze Anwendungen oder einzelne Erweiterungen in sog. Plugins gekapselt. Ein Plugin ist eine Komponente, welche eine Funktionalität im Eclipse-Kontext anbietet, ähnlich einem Java-Package[2]. Ein Erweiterungspunkt ist im Prinzip eine von der Eclipse Plattform bereitgestellte Tabelle, in der sich Plugins in einer Zeile eintragen können. Eine solche Tabelle wird als Registry bezeichnet. Die Inhalte der Registry können durch die eigene Anwendung von der Plattform abgerufen werden.

Um erweiterbar zu sein, muss das eigene Plugin einen Erweiterungspunkt definieren. Da die Eclipse-Plattform für jeden Erweiterungspunkt eine eigene Tabelle bereitstellt, ist die Anzahl und der Wertebereich der Spalten durch das definierende Plugin zu spezifieren. Außerdem ist anzugeben, ob die Werte in den Spalten obligatorisch oder optional sind. Damit zwei Plugins zur Laufzeit miteinander interagieren können, ist zum Beispiel immer die Definition einer Schnittstelle erforderlich, auf die das Extension-Object des erweiterenden Plugins vom eigenen Plugin gecastet werden kann. Im Dateisystem wird diese Spezifikation durch eine XML-Datei abgebildet.

Um in die Tabelle eines Erweiterungspunktes aufgenommen zu werden, muss ein Plugin diesen “bedienen”. Bedienen heißt, dass es in seinen eigenen XML-Metadaten (Datei “plugin.xml”) mindestens die obligatorischen Werte der Tabellenspalten speichern muss. Beim Start der Eclipse-Plattform erzeugt diese für jeden Erweiterungspunkt die entsprechende Tabelle, prüft alle Plugins auf Erweiterungen für diesen Erweiterungspunkt und trägt diese gegebenenfalls in die Tabelle ein. Zum Initialisierungszeitpunkt der Plugins steht so eine “gefüllte” Erweiterungstabelle zur Verfügung, deren Inhalte durch das eigene Plugin von der Plattform abgerufen werden können.

[1] Die Bibliothek kann von der Internetseite http://developer.yahoo.com/search/ bezogen werden.
[2] vgl. [1] Kapitel 1 - Introduction

C.2 Adapterklasse “YahooService”

Die Klasse “YahooService” adaptiert die Schnittstelle “SearchService” des Assistenten auf die Funktionalität der Yahoo-Bibliothek. Die Klassen “SearchClient”, “WebSearchRequest” und “WebSearchResults” stammen aus dieser Bibliothek. Im folgenden Listing ist exemplarisch die Methode “executeQuery” angegeben, da diese den Kommunikationsprozess mit der Suchmaschine Yahoo kapselt.

```
1  public class Yahoo implements SearchService {
2   ...
3
4   public List<Hit> executeQuery(final String query,
5     final int maxResultNumber) {
6    // Create the search client.
7    SearchClient client = new SearchClient(APP_ID);
8    // Create the web search request.
9    WebSearchRequest request = new WebSearchRequest(query);
10   request.setResults(maxResultNumber);
11   request.setCountry("de");
12   request.setLanguage("de");
13   request.setParameter("region", "de");
14   List<Hit> result = new ArrayList<Hit>();
15   available = true;
16
17   try {
18    // Execute the search.
19    WebSearchResults results = client.webSearch(request);
20    // Iterate over the results.
21    for (int i = 0; i < results.listResults().length &&
22     available; i++) {
23     WebSearchResult result = results.listResults()[i];
24     result.add(new YahooHit(result,this));
25    }
26   } catch (com.yahoo.search.SearchException e) {
27    e.printStackTrace();
28    available = false;
29   } catch (IOException e) {
30    available = false;
31   }
32   return result;
33  }
34
35  ...
36 }
```

An dieser Stelle soll kurz skizziert werden, wie die Methode arbeitet. Die Suchmaschine Yahoo stellt einen Webservice bereit, der in Form einer REST-Architektur (REpresentational State Transfer) aufgebaut worden ist. REST ist kein ein Produkt oder Standard, sondern ein Architekturstil. Im REST-Stil entworfene Webservices kommunizieren über das HTTP-Protokoll (Hyper Text Transfer Protocol). Sie wenden die HTTP-Operationen GET, PUT, POST und DELETE auf über URLs adressierte Ressourcen an. Über diese Operationen können Nachrichten an die Ressourcen gesendet werden. Eine Nachricht kann prinzipiell in jedem beliebigen Dokumenttyp formatiert sein[3].

Die Methode “executeQuery()” instanziiert zuerst ein Objekt der Klasse “SearchClient”. Dieses Objekt ist verantwortlich für die Kommunikation mit dem Yahoo-Webservice. Es transformiert die gestellten Suchanfragen in eine XML-Nachricht und sendet diese über eine GET-Operation an den Webservice. Ebenso transformiert es die erhaltenen XML-Antworten vom Webservice in Java-Objekte. Der Konstruktor der Klasse erhält als Parameter eine Applikations-ID. Diese ID wird von Yahoo für vergeben und identifiziert das eigene Programm (nicht den Benutzer).

Anschließend wird die übergebene String-Suchanfrage “query” über eine Instanz der Klasse “WebSearchRequest” in Yahoo-Syntax abgebildet. An diesem Objekt wird ebenfalls zu adressierende Yahoo-Index spezifiziert. Die Methode “executeQuery()” sucht nur im deutschsprachigen Index.

Anschließend wird die Anfrage an den Webservice gesendet und die Treffer empfangen. Die Schnittstelle “WebSearchResults” repräsentiert die von Yahoo gelieferte Trefferliste. Ein Treffer wird durch ein Objekt die Schnittstelle “WebSearchResult” in Java abgebildet. Diese Treffer-Objekte werden von Instanzen der Klasse “YahooHit” adaptiert und an den Assistenten zurückgegeben.

[3] Für detaillierte Informationen zum Architektur-Stil REST wird auf die Dissertation von Roy Thomas Fielding unter http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm verwiesen.

C.3 Adapterklasse “YahooHit”

Nachfolgend ist die Implementierung der Adapterklasse für einen Yahoo-Treffer (Objekt vom Typ WebSearchResult) auszugsweise angegeben.

```
37 public class YahooHit implements Hit {
38  private WebSearchResult _result = null;
39  private SearchService _s = null;
40
41  public YahooHit(final WebSearchResult result, SearchService s) {
42   _result = result;
43   _s = s;
44  }
45  ...
46  public Set<Pair<String,Double>> getContents() {
47   Set<Pair<String, Double>> result =
48     new HashSet<Pair<String, Double>>();
49   int dotIndex = _result.getUrl().lastIndexOf('.');
50   //Extract file-extension
51   String fileExt = result.getUrl().substring(dotIndex);
52   String mimeType = "";
53   InputStream in = null;
54
55   try {
56    mimeType = MimeTypeService.getInstance().getMimeType(fileExt);
57    in = new URL(_result.getUrl()).openStream();
58    result = DocumentReaderService.getInstance().
59      readDocumentContents(mimeType,in);
60   } catch (Exception e) {
61    result.clear();
62    result.add(new Pair<String,Double>("",10.0));
63   }
64   return result;
65  }
66 }
```

In der Methode “getContents()” die gewichteten Inhalte des Treffers ausgelesen. Dazu wird zuerst anhand seiner Dateiendung der MIME-Typ des Treffers über die Singleton-Instanz der Klasse “MimeTypeService” bestimmt. Anschließend wird über die Klasse “java.net.URL” ein “InputStream”-Objekt zur durch den Treffer referenzierten Ressource erzeugt. Abschließend extrahiert die Singleton-Instanz der Klasse “DocumentReaderService” die gewichteten Inhalte aus dem Treffer. Diese werden als Methodenresultat zurückgeliefert.

C.4 XSD-Spezifikation für Eclipse-Erweiterungspunkt

Die Geschäftslogik des Assistenten ist in einem eigenen Eclipse-Plugin entwickelt worden. Dieses Plugin definiert einen Erweiterungspunkt, an dem sich weitere Suchmaschinen anmelden können. Hinterlegt ist die Spezifikation dieses Erweiterungspunktes in XSD-Notation (XML Schema Definition) in der Datei “schema/searchservices.exsd” im Plugin-Verzeichnis der Assistentengeschäftslogik”.

Nachfolgend ist die exakte Spezifikation angegeben:

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Schema file written by PDE -->
<schema targetNamespace="org.lassitools.assistantservice">
<annotation>
 <appInfo>
  <meta.schema plugin="org.lassitools.assistantservice"
    id="searchServices" name="searchServices"/>
 </appInfo>
 <documentation>
  Via this Extensionpoint it is possible to add
  further search-engines to the LAssi-Searchassistant.
 </documentation>
</annotation>
<element name="extension">
 <complexType>
  <sequence><element ref="searchService"/></sequence>
  <attribute name="point" type="string" use="required"/>
  <attribute name="id" type="string"/>
  <attribute name="name" type="string">
  <annotation>
   <appInfo>
    <meta.attribute translatable="true"/>
   </appInfo>
   </annotation>
  </attribute>
 </complexType>
</element>
<element name="searchService">
 <annotation>
  <appInfo>
   <meta.element labelAttribute="name"/>
  </appInfo>
 </annotation>
 <complexType>
  <attribute name="id" type="string" use="required">
```

```
102     <annotation>
103       <documentation>
104         The id of the search-engine.
105       </documentation>
106     </annotation>
107   </attribute>
108   <attribute name="class" type="string" use="required">
109     <annotation>
110       <documentation>
111         The adapterclass of the search-engine.
112       </documentation>
113       <appInfo>
114         <meta.attribute kind="java"
115           basedOn="org.lassitools.assistantservice.
116 ␣␣␣␣␣␣␣␣searchservices.SearchService"/>
117       </appInfo>
118     </annotation>
119   </attribute>
120  </complexType>
121 </element>
122 </schema>
```

Die Spezifikation verlangt von einer, diesen Erweiterungspunkt bedienenden Suchmaschine, die Angabe einer ID und einer Klasse, die die Schnittstelle “SearchService” implementiert. Die ID dient als Primärschlüssle in der Erweiterungspunkttabelle des Eclipse-Plattform. Die Adapterklasse der Suchmaschine muss sich ebenfalls in einem Eclipse-Plugin befinden, denn nur Plugins können die Erweiterungspunkte von anderen Plugins bedienen. In der Datei “plugin.xml” des Plugins der Suchmaschine sieht die Bedienung des Erweiterungspunktes wie folgt aus:

```
123 <?xml version="1.0" encoding="UTF-8"?>
124 <?eclipse version="3.0"?>
125 <plugin>
126  <extension point="org.lassitools.agentservice.searchServices">
127   <searchService
128     class="org.lassitools.assistantservice.services.Yahoo"
129     id="org.lassitools.assistantservice.Yahoo"/>
130  </extension>
131 </plugin>
```

D Schülerfragebogen zur Bewertung der Assistentenleistung

ss

E Abkürzungsverzeichnis

WWW	World Wide Web
HTML	Hyper Text Markup Language
XML	Extensible Markup Language
PDF	Portable Document Format
RTF	Rich Text Format
HTTP	Hyper Text Transfer Protocol
MIME	Multipurpose Internet Mail Extensions
URL	Uniform Resource Locator
UML	Unified Modeling Language
z.B.	zum Beispiel

Literaturverzeichnis

[1] A. Bolour. Notes on the eclipse plug-in architecture, 2003. Verfügbar unter http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine, 1998. Verfügbar unter http://wx.bsjh.tcc.edu.tw/~t2003013/wiki/images/8/8f/Anatomy.pdf.

[3] R. Ferber. *Information Retrieval*. dpunkt.verlag GmbH, 2003. Verfügbar unter http://information-retrieval.de.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Entwurfsmuster (Deutsche Übersetzung)*. Addison-Wesley Publishing Company, 2004.

[5] O. Gospodnetić and E. Hatcher. *Lucene in Action*. Manning Publications Co., 2005.

[6] H. Kaufmann and H. Pape. *Clusteranalyse*, pages 437–536. Walter de Gryter und Co., 1996. Erschienen in Ludwig Fahrmeir und Alfred Hamerle und Gerhard Tutz (Hrsg.): Multivariate statistische Verfahren.

[7] L. Papula. *Mathematik für Ingenieure und Naturwissenschaftler - Band 1*. Friedr. Vieweg und Sohn Verlagsgesellschaft mbH, 2001.

[8] M. F. Porter. An algorithm for suffix stripping. *Program, 14 no. 3*, pages 130–137, July 1980. Verfügbar unter http://www.tartarus.org/~martin/PorterStemmer/def.txt.

[9] T. A. Runkler. *Information Mining - Methoden, Algorithmen und Anwendungen intelligenter Datenanalyse*. Friedr. Vieweg und Sohn Verlagsgesellschaft mbH, 2000.

[10] G. Salton and M. McGill. *Information Retrieval - Grundlegendes für Informationswissenschaftlicher (Deutsche Übersetzung)*. McGraw - Hill Book Company GmbH, 1983.

[11] S. J. R. und Peter Norvig. *Künstliche Intelligenz - ein moderner Ansatz (Deutsche Übersetzung)*. Addison-Wesley Verlag, 2004.

[12] S. M. Weiss, N. Indurkhya, T. Zhang, and F. J. Damerau. *Text Mining - Predictive Methods for Analyzing Unstructured Information*. Springer Science + Business Media, Inc., 2004.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Buxtehude, den 21.02.2007 *Jan Christian Krause*